SQL Server 2016

New Feature Preview

2015-10-10

@RileyMajor

PASS

PASS SQL saturday

#453 | MINNESOTA 2015

SQL Saturday #453

- Thank you Sponsors!
  - Please visit the sponsors during the vendor break from 2:45 – 3:15 and enter their end-of-day raffles
- Event After Party
  - Dave and Buster’s in Southdale Center. 3rd floor by Macy’s starting at 6:15
- Want More Free Training?
  - PASSMN meets the 3rd Tuesday of every month. https://mnssug.org/

SQL Saturday #453

- Twitter
  - @PASSMN
  - #SQLSatMN

Presentation Overview

- Brief History of SQL Server
- New Features in 2016
  - JSON
  - Temporal Tables
  - Interactive Query Plans
- Getting Started with 2016
- Future Thoughts
- Bio

Birth of SQL Server

- Started with Ashton Tate (dBase) and Sybase
- Version 1 Shipped in 1989
- Ditched Ashton Tate with dBase IV failure
- Version 1.1 Shipped in 1990 – Sybase adds support for Windows
- Version 4.2 Shipped in 1992 – First significant Microsoft involvement

SQL Server on Windows

- Parted with Sybase in 1993
- Focused on Windows only.
- Version 6 shipped in 1996 – No further Sybase involvement.
- Version 7 shipped in 1998 – DTS Packages, OLAP tools.
- SQL Server 2000 – User-Defined Functions, Reporting and Analysis Services

SQL Server Grows Up

- SQL Server 2005 (v9)
  - Sybase Code “completely rewritten”.
  - SSMS
  - XML
  - TRY/ CATCH
  - APPLY Operator, Common Table Expressions
  - varchar(max)
  - OUTPUT Clause
  - PIVOT, Ranking Window Functions (ROW_NUBER; OVER)
  - Encryption, Hashing
  - Dynamic Management Views (DMVs), DDL Triggers, Query Notifications
  - Database Mirroring, Service Broker, SSIS, Database Mail, CLR
  - Database Snapshots*, Table Partitioning*, Online Index Operations*, Mirrored Backups*

The Hits Keep Coming

- SQL Server 2008 (v10)
  - Date and Time Data Types
  - VALUES (Table Value Constructor)
  - Extended Events
  - Table-Valued Parameters
  - MERGE
  - HIERARCHYID
  - Geography and Geometry Spatial Data Types
  - Filtered Indexes
  - Grouping Sets
  - Change Data Capture*
  - Compression*
  - Resource Governor*
  - Transparent Data Encryption*

There is no SQL Server 2010

- SQL Server 2008 R2 (v10.5)
  - Compression for Standard (Log/Backup only)
  - BI Tooling (Reporting Services, PowerPivot)
  - Master Data Services*
  - StreamInsight**
- Well, it was only a point release.
- Last Chance for Slot-based Licensing

That’s a little better…

- SQL Server 2012 (v11)
  - TRY_PARSE, PARSE, TRY_CONVERT
  - FORMAT
  - CHOOSE, IIF
  - THROW
  - OFFSET / FETCH
  - Window Function Enhancements (LAG, LEAD)
  - User-Defined Server Security Roles
  - Contained Databases
  - AlwaysOn*
  - Columnstore Indexes*
  - Analysis Services Tabular Model*

DENY on T_SQL

- SQL Server 2014 (v12)
  - Backup to Azure
  - Delayed Durability
  - Encrypted Backups
  - Buffer Pool Extension (SSD Cache)
  - Updatable Columnstore Indexes*
  - In-Memory OLTP (Hekaton)*

The New and Shiny

- SQL Server 2016
  - Query Store
  - Dynamic Data Masking
  - JSON
  - Temporal Tables
  - Live Query Statistics
  - Always Encrypted
  - Multiple Secondary Readers for Load Balancing
  - R Algorithms in SQL
  - Stretch Database (Auto-Archive to Azure)
  - Row-Level Security
  - Automatic TempDB Optimization
  - PolyBase (T-SQL for Hadoop)

Cha-ching!

- Query Store – store ALL the query plans!
- Your plan cache is only so big, but your disks are bigger.
- Why recompute when you can reload?
- Reboot? Would you like me to warm that cache up for you?
- Don't like this new plan? Return it for a full refund!

<REDACTED>

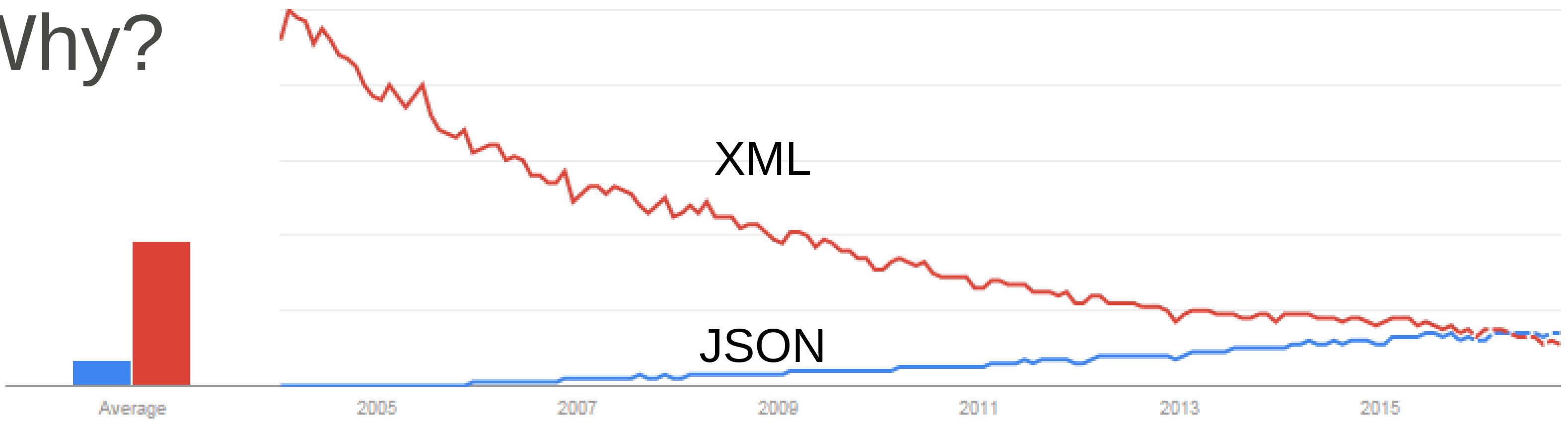
- Dynamic Data Masking hides sensitive information (such as personally-identifiable information– PII) from normal users.
- There are a variety of masking functions which turn text into XXXX, dates into 2000-01-01, etc.
- UNMASK permissions are required to see real data.

Hold this but don’t look inside!

- Always Encrypted hides data even from DBAs.
- Encryption keys live with application.
- Performance issues slightly mitigated with repeatable encryption, but that weakens security.
- Suggested for use with external DBA resources.

JSON

- Why?



- What?

```
[
	{
		"MyField1": "MyValue",
		"MyField2": 123.45
	},
	{
		"MyField1": "MyValue2",
		"MyField2": 345.67
	}
]
```

JSON vs XML – 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

JSON vs XML – Production (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

```
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 vs XML – Production (Path)

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
	},
	{
		"OrderID":1,
		"OrderDate":"2015-10-10T00:00:00",
		"ProductID":"Wheels",
		"Qty":4
	},
	{
		"OrderID":2,
		"OrderDate":"2015-10-09T00:00:00",
		"ProductID":"Ball",
		"Qty":10
	}
]
```

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>
<row>
	<OrderID>1</OrderID>
	<OrderDate>2015-10-10T00:00:00</OrderDate>
	<ProductID>Wheels</ProductID>
	<Qty>4</Qty>
</row>
<row>
	<OrderID>2</OrderID>
	<OrderDate>2015-10-09T00:00:00</OrderDate>
	<ProductID>Ball</ProductID>
	<Qty>10</Qty>
</row>
```

JSON vs XML – Production (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

```
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 vs XML – Production (Auto)

JSON

```
[
{
        "OrderID":1,
        "OrderDate":"2015-10-10T00:00:00",
        "OrderDetails":
        [
                { "ProductID":"Bike", "Qty":2 },
                { "ProductID":"Helmet", "Qty":2 },
                { "ProductID":"Wheels",  "Qty":4 }
        ]
},
{
        "OrderID":2,
        "OrderDate":"2015-10-09T00:00:00",
        "OrderDetails":
        [
                { "ProductID":"Ball", "Qty":10 }
        ]
}
]
```

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>
<Orders OrderID="2" OrderDate="2015-10-09T00:00:00">
        <OrderDetails ProductID="Ball" Qty="10" />
</Orders>
```

JSON vs XML – Path with Nesting

JSON

```
SELECT
	Orders.OrderID,
	Orders.OrderDate,
	(
		SELECT
			OrderDetails.ProductID,
			OrderDetails.Qty
		FROM	@OrderDetails AS
			OrderDetails
		WHERE	Orders.OrderID =
			OrderDetails.OrderID
		FOR JSON PATH
	) AS OrderDetails
FROM			@Orders Orders
FOR			JSON PATH,
			ROOT('Orders');
```

XML

```
SELECT
	Orders.OrderID,
	Orders.OrderDate,
	(
		SELECT
			OrderDetails.ProductID,
			OrderDetails.Qty
		FROM	@OrderDetails AS
			OrderDetails
		WHERE	Orders.OrderID =
			OrderDetails.OrderID
		FOR XML PATH('OrderDetail'), TYPE
	) AS OrderDetails
FROM			@Orders Orders
FOR			XML PATH('Order'),
			ROOT('Orders');
```

JSON vs XML – Path with Nesting

JSON

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

XML

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

Unholy Unions

XML in JSON

```
{
	"UnholyUnion":
		"<DataList DataElement=\"Yes, you can put XML in JSON!\"\/><DataList DataElement=\"But why would you do this?\"\/>"
}
```

JSON in XML

```
<row>
	<UnholyUnion>
		[{"DataElement":"Yes, you can put JSON in XML!"},{"DataElement":"But why would you do this?"}]
	</UnholyUnion>
</row>
```

Look ma, no tags!

XML

```
SELECT
	'Test'
FOR XML PATH('');
```

Results:

```
Test
```

JSON

```
SELECT
	'Test'
FOR JSON PATH;
```

Results:

```
Msg 13605, Level 16, State 1, Line 1
Unnamed tables cannot be used as JSON identifiers as well as unnamed columns cannot be used as key names. Add alias to the unnamed column/table.
```

You’re just not my type.

- XML is a data type.
- JSON is *not* a data type. Use NVARCHAR.
  - 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?

Nettlesome Nesting

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.\"}"}
```

Nettlesome Nesting - Workaround

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

Results:

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

Well is it or isn’t it?

- 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.

Is it “rows” or “records”?

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 varchar(10) '.');
```

nodes

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

SELECT
	a.value('.','varchar(10)')
FROM	@x.nodes('/x/a') AS x(a);
```

But I haven’t prepared!

- There is no nodes-style syntax for JSON.
- OPENJSON has similar syntax to OPENXML.
- No prepare statement is needed.
  - Work in user-defined function?
  - Multiple in single SQL statement?
  - Performance?

OPENJSON

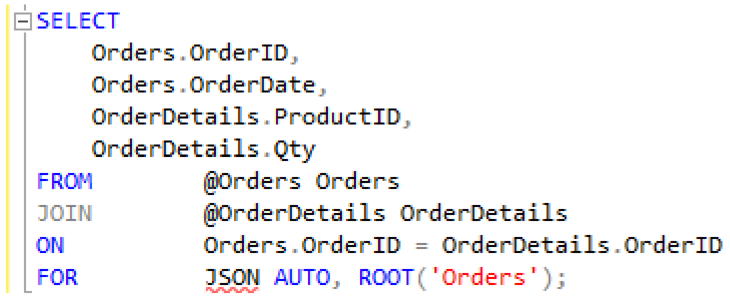
```
DECLARE @j nvarchar(max) = '{"Orders": [
{"OrderID":1, "OrderDate": "2015-10-10"},
{"OrderID":2, "OrderDate": "2015-10-09"}]}';

SELECT
	OrderID, OrderDate
FROM OPENJSON (@j, '$.Orders')
WITH
(
	OrderID bigint,
	OrderDate datetime
) AS OrdersArray;
```

Third time's the charm.

- FOR JSON works in CTP 2.4.
- We have to wait until CTP 3 for:
  - OPENJSON()
  - JSON_VALUE()
  - ISJSON()

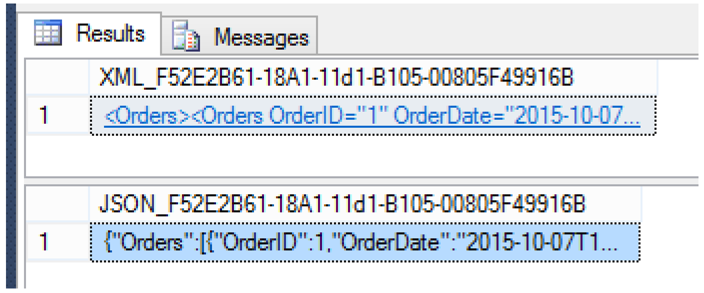
Jason who?

- SSMS sees JSON as syntax error.



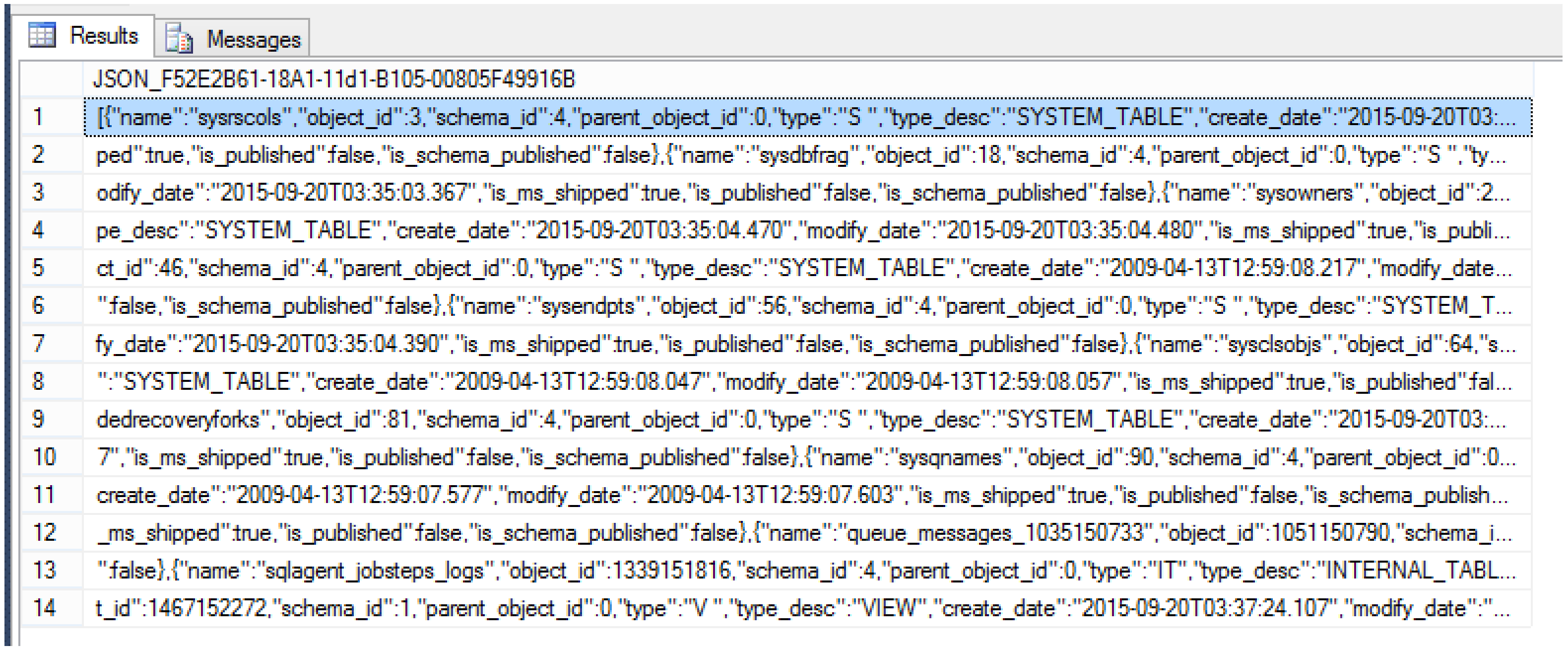
```
SELECT
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.ProductID,
    OrderDetails.Qty
FROM        @Orders Orders
JOIN        @OrderDetails OrderDetails
ON          Orders.OrderID = OrderDetails.OrderID
FOR         JSON AUTO, ROOT('Orders');
```

- SSMS has no formatting or special handling for JSON results.



```
Results   Messages

   XML_F52E2B61-18A1-11d1-B105-00805F49916B
1  <Orders><Orders OrderID="1" OrderDate="2015-10-07...

   JSON_F52E2B61-18A1-11d1-B105-00805F49916B
1  {"Orders":[{"OrderID":1,"OrderDate":"2015-10-07T1...
```

Yuck.



Results | Messages

```
     JSON_F52E2B61-18A1-11d1-B105-00805F49916B
1    [{"name":"sysrscols","object_id":3,"schema_id":4,"parent_object_id":0,"type":"S ","type_desc":"SYSTEM_TABLE","create_date":"2015-09-20T03:...
2    ped":true,"is_published":false,"is_schema_published":false},{"name":"sysdbfrag","object_id":18,"schema_id":4,"parent_object_id":0,"type":"S ","ty...
3    odify_date":"2015-09-20T03:35:03.367","is_ms_shipped":true,"is_published":false,"is_schema_published":false},{"name":"sysowners","object_id":2...
4    pe_desc":"SYSTEM_TABLE","create_date":"2015-09-20T03:35:04.470","modify_date":"2015-09-20T03:35:04.480","is_ms_shipped":true,"is_publi...
5    ct_id":46,"schema_id":4,"parent_object_id":0,"type":"S ","type_desc":"SYSTEM_TABLE","create_date":"2009-04-13T12:59:08.217","modify_date...
6    ":false,"is_schema_published":false},{"name":"sysendpts","object_id":56,"schema_id":4,"parent_object_id":0,"type":"S ","type_desc":"SYSTEM_T...
7    fy_date":"2015-09-20T03:35:04.390","is_ms_shipped":true,"is_published":false,"is_schema_published":false},{"name":"sysclsobjs","object_id":64,"s...
8    ":"SYSTEM_TABLE","create_date":"2009-04-13T12:59:08.047","modify_date":"2009-04-13T12:59:08.057","is_ms_shipped":true,"is_published":fal...
9    dedrecoveryforks","object_id":81,"schema_id":4,"parent_object_id":0,"type":"S ","type_desc":"SYSTEM_TABLE","create_date":"2015-09-20T03:...
10   7","is_ms_shipped":true,"is_published":false,"is_schema_published":false},{"name":"sysqnames","object_id":90,"schema_id":4,"parent_object_id":0...
11   create_date":"2009-04-13T12:59:07.577","modify_date":"2009-04-13T12:59:07.603","is_ms_shipped":true,"is_published":false,"is_schema_publish...
12   _ms_shipped":true,"is_published":false,"is_schema_published":false},{"name":"queue_messages_1035150733","object_id":1051150790,"schema_i...
13   ":false},{"name":"sqlagent_jobsteps_logs","object_id":1339151816,"schema_id":4,"parent_object_id":0,"type":"IT","type_desc":"INTERNAL_TABL...
14   t_id":1467152272,"schema_id":1,"parent_object_id":0,"type":"V ","type_desc":"VIEW","create_date":"2015-09-20T03:37:24.107","modify_date":"...
```

JSON Summary

- FOR JSON
  - Works pretty much like FOR XML.
- OPENJSON
  - Similar to OPENXML, but no need for separate “preparation” step (sp_xml_preparedocument).
- ISJSON(@JSON)
  - Similar to TRY_CONVERT(xml, '<x>xml</x>')
- JSON_VALUE(@JSON, '$.Order.Qty')
  - Similar to @XML.value

Temporal Tables

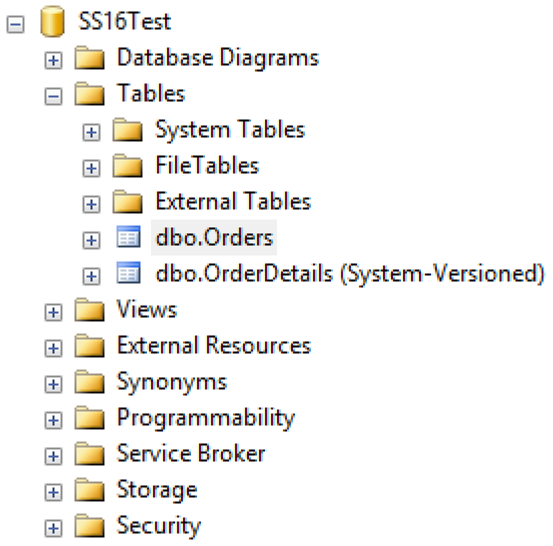
- Like Apple's Time Machine, but for your tables.
- Can operate entirely transparently to existing applications.
- Main table gets two extra time stamp fields.
- History table:
  - Has same schema and stores old information.
  - Uses compression by default.
  - Can be queried directly.
  - Microsoft suggests using Azure stretch table.

Temporal Tables – Basic Syntax

```
CREATE TABLE OrderDetails
    (
        OrderDetailID bigint IDENTITY PRIMARY KEY,
        OrderID bigint,
        ProductID varchar(50),
        Qty int,
        EffectiveStart datetime2
            GENERATED ALWAYS AS ROW START NOT NULL,
        EffectiveStop datetime2
            GENERATED ALWAYS AS ROW END NOT NULL,
        PERIOD FOR SYSTEM_TIME (EffectiveStart, EffectiveStop)
    )
    WITH (SYSTEM_VERSIONING =
        ON (HISTORY_TABLE = dbo.OrderDetailsHistory));
```

Temporal Tables – Time Travelling

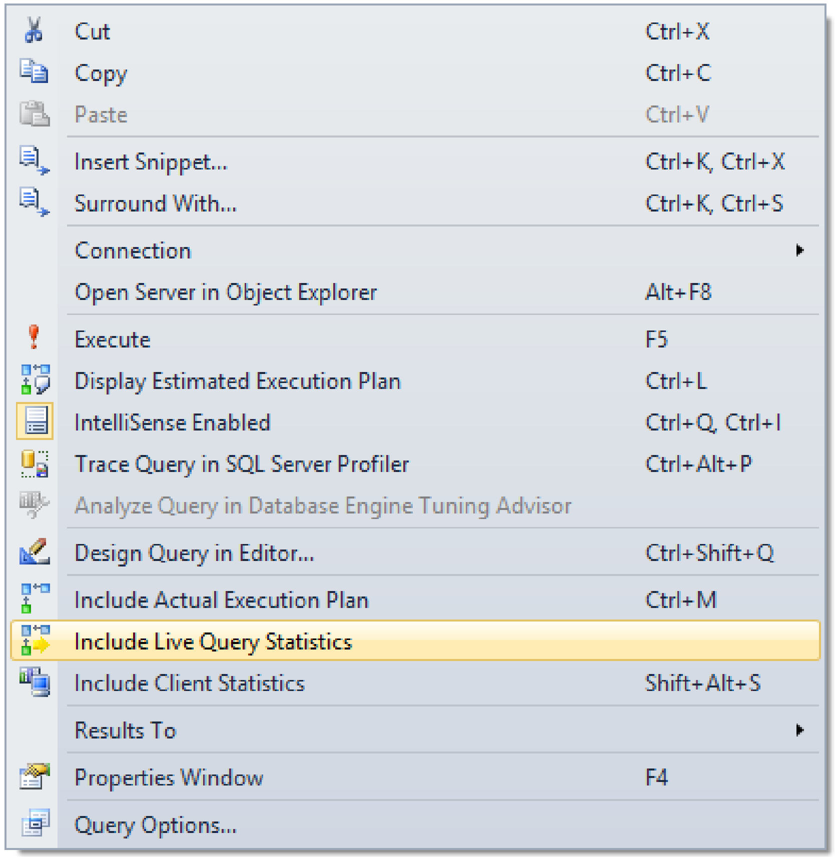
```
SELECT    *
FROM      OrderDetails
   FOR
   SYSTEM_TIME
   AS OF @dt
ORDER BY
   OrderDetailID;
```



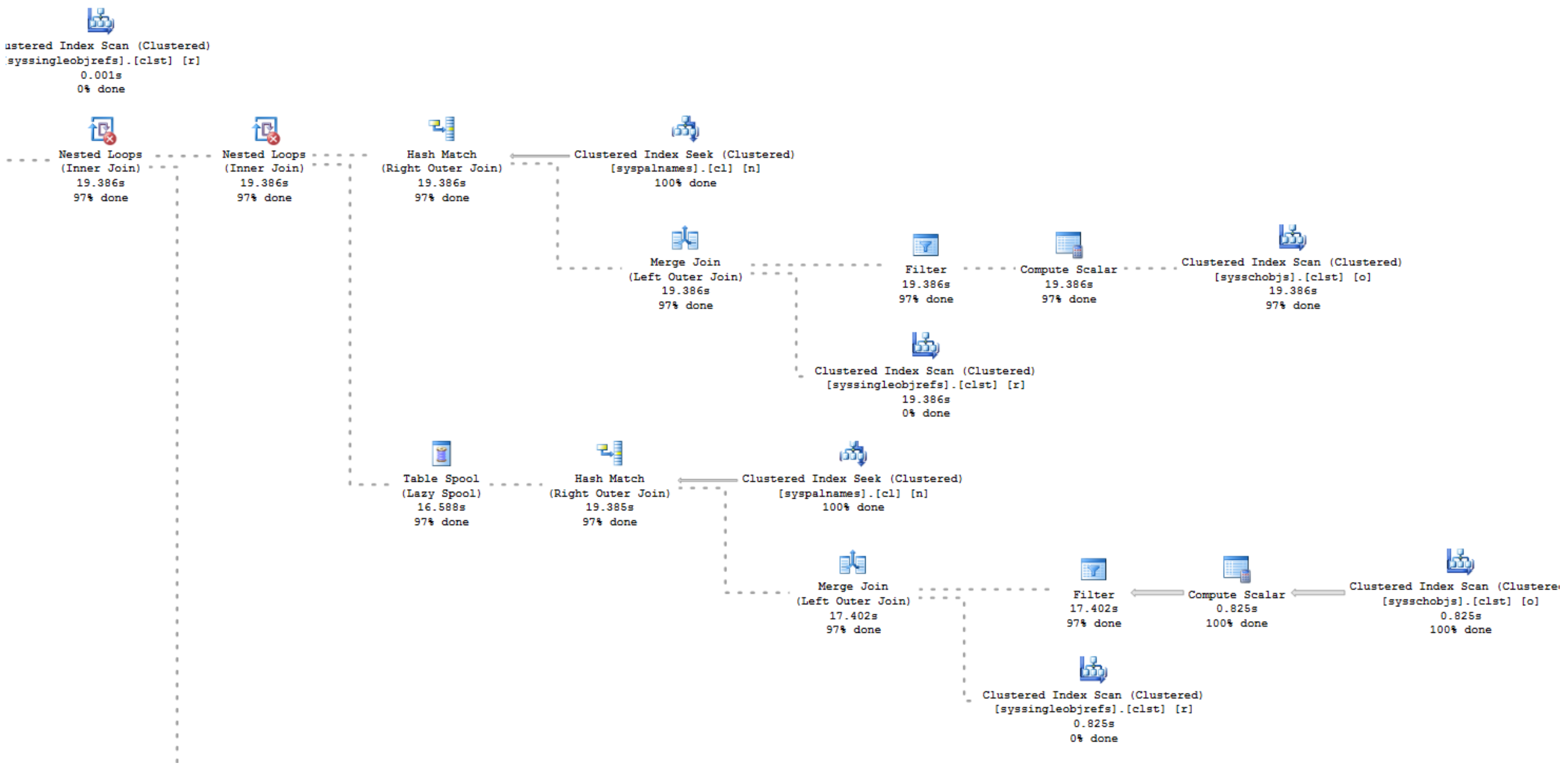
```
⊟ SS16Test
   ⊞ Database Diagrams
   ⊟ Tables
      ⊞ System Tables
      ⊞ FileTables
      ⊞ External Tables
      ⊞ dbo.Orders
      ⊞ dbo.OrderDetails (System-Versioned)
   ⊞ Views
   ⊞ External Resources
   ⊞ Synonyms
   ⊞ Programmability
   ⊞ Service Broker
   ⊞ Storage
   ⊞ Security
```

Live Query Statistics

- Boom!
- Wow.
- “The performance impact of turning this on, on a production server, could be significant.” – Russ Thomas
- ☹


Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Insert Snippet... Ctrl+K, Ctrl+X
Surround With... Ctrl+K, Ctrl+S
Connection
Open Server in Object Explorer Alt+F8
Execute F5
Display Estimated Execution Plan Ctrl+L
IntelliSense Enabled Ctrl+Q, Ctrl+I
Trace Query in SQL Server Profiler Ctrl+Alt+P
Analyze Query in Database Engine Tuning Advisor
Design Query in Editor... Ctrl+Shift+Q
Include Actual Execution Plan Ctrl+M
Include Live Query Statistics
Include Client Statistics Shift+Alt+S
Results To
Properties Window F4
Query Options...

Feast your eyes…



Can I play too?

- Download and Install
  - Best with Virtualization.
  - 2 gig for SQL Server download. 4 gig for Windows.
  - This can take several hours to download and set up.
  - Free for evaluation for 120 days.
- TechNet Virtual Labs
  - Free, but limited options.
- Azure Virtual Machine
  - SQL Server 2016 Pre-Installed
- Azure SQL Database
  - Eventually

Path of Most Resistance

- You need a VM host if you don’t want to mess with multiple SQL Server and SSMS versions on your main box.
  - VirtualBox is free and cross-platform.
- You need an OS.
  - Windows 10 isn’t supported on VirtualBox (yet). You can get free evaluation version of Windows 8.1; lasts for 120 days.
  - If you have a spare license, you can download normal Windows 8.1 ISO. But you can’t use that for the evaluation version.

Give me more. I can take it.

- You need the SQL Server 2016 CTP bits.
  - ISO – Can burn to DVD or mount with software. In retrospect, this is the best option.
  - EXE + BOX (compressed file like a CAB or ZIP) – Can just run installation.
    - However, you need space to host uncompressed and compressed files *in* the VM.
  - Azure – Mostly useful if you are testing in an Azure VM.
- Evaluation lasts for 180 days.

I’ve got all day.

- Use VM software to make blank VM.
- Mount ISO image using VM software.
- Install Windows.
  - Enable .NET 3.5.
  - Install Java (if you want Polybase).
  - Install VM tools (to access files outside VM).
- Install SQL Server.
- So many updates!
- Plenty of reboots, too.

Going for Broke

- Which will require Enterprise edition?
- Look at past enhancements:
  - 2005 was almost all Standard Edition.
  - Since then, headline features are Enterprise.
  - But T-SQL features usually Standard.
- “SQL Server 2014 Standard Edition Sucks, and It’s All Your Fault” – Brent Ozar, 2013-07-29

Place your bets…

- Enterprise
  - PolyBase, R Algorithms in SQL
    - Big Data is Big Bucks
  - Dynamic Data Masking
    - Compliance is Big Bucks
  - Always Encrypted & Row-Level Security
    - TDE and Fine-Grained Auditing
  - Multiple Secondary Readers
    - AlwaysOn AGs
  - Temporal Tables
    - Change Data Capture, Uses Compression

The Crumbs

- Standard
  - JSON
    - XML
  - Stretch Database
    - Previous Azure Support
  - Automatic TempDB Optimization
    - Buffer Pool & Delayed Durability
  - Live Query Statistics & Query Store
    - Most “RDBMS Manageability” is in Standard

Say Goodbye

- SQL Server 2016 will discontinue and/or break some features, but they are “to be determined”.

Steer Clear

- Discontinued in next version
  - SET ROWCOUNT for data modification.
    - Use TOP instead.
  - Result Sets from Triggers
  - Remote Servers
    - Use Linked Servers instead.
- Still kicking:
  - Database Mirroring
  - Omitting Semicolons

The People Have Spoken

- JSON Support is #1 Connect Item
  - 1070 Up-Votes
- But…
  - CREATE OR REPLACE
  - 7th Most Popular (442 Up-Votes)
  - Created in March of 2005

Shut Up and Take My Money

“While this has also been hotly requested, I’d caution you to **be careful what you ask for**. Users demanded XML support inside SQL Server, and then proceeded to use SQL Server as an XML query engine, sending CPU through the roof. **SQL Server is one of the world’s most expensive application servers.**”

– Brent Ozar, 2015-05-04

In the year 2000…

- SQL Server 2018?
- Brent Ozar: VMs mean 2005 & 2008 4eva
- Other popular Connect items:
  - Better Error Info for “String… truncated”
  - Error Table for Big Insert/Update Failures
  - Inline Scalar UDFs
  - Full Regex for LIKE, etc.
  - Built-in Tally Table (Table of Numbers)
  - Read/Write Table-Valued Parameters

Riley Major

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

Evaluations Please

- Remember to fill out your online evaluations for the event and any sessions you have attended. They will be online until 10/17/15.

http://www.sqlsaturday.com/453/eventeval.aspx
http://www.sqlsaturday.com/453/sessions/sessionevaluation.aspx