


# Cool SQL Server Features Every Developer Should Know About

David Berry  
@DavidCBerry13  
September 9, 2017



# Sample Database and Code

 [Features](#) [Business](#) [Explore](#) [Pricing](#)

This repository

[Sign in](#) or [Sign up](#)

DavidCBerry13 / CoolSqlServerFeatures

Watch1

★ Star0

🍴 Fork0

[Code](#) [Issues0](#) [Pull requests0](#) [Projects0](#) [Pulse](#) [Graphs](#)

Sample database and queries for my talk about Cool SQL Server Features


3 commits

1 branch

0 releases

1 contributor

Branch: master [New pull request](#) [Find file](#) [Clone or download](#)

 DavidCBerry13 Added SQL for MERGE statement, example window function queries Latest commit a6dc595 17 seconds ago

JSONFeatures	Initial revision	a day ago
MergeStatement	Added SQL for MERGE statement, example window function queries	17 seconds ago
TemporalTables	Initial revision	a day ago
WindowFunctions	Added SQL for MERGE statement, example window function queries	17 seconds ago
.gitattributes	Added .gitattributes & .gitignore files	a day ago
.gitignore	Added .gitattributes & .gitignore files	a day ago

<https://github.com/DavidCBerry13/CoolSqlServerFeatures/>

We all use SQL Server every day...  
...we might as well be good at it.

# Agenda

Temporal  
Tables

JSON  
Support

Window  
Functions

# Temporal Tables

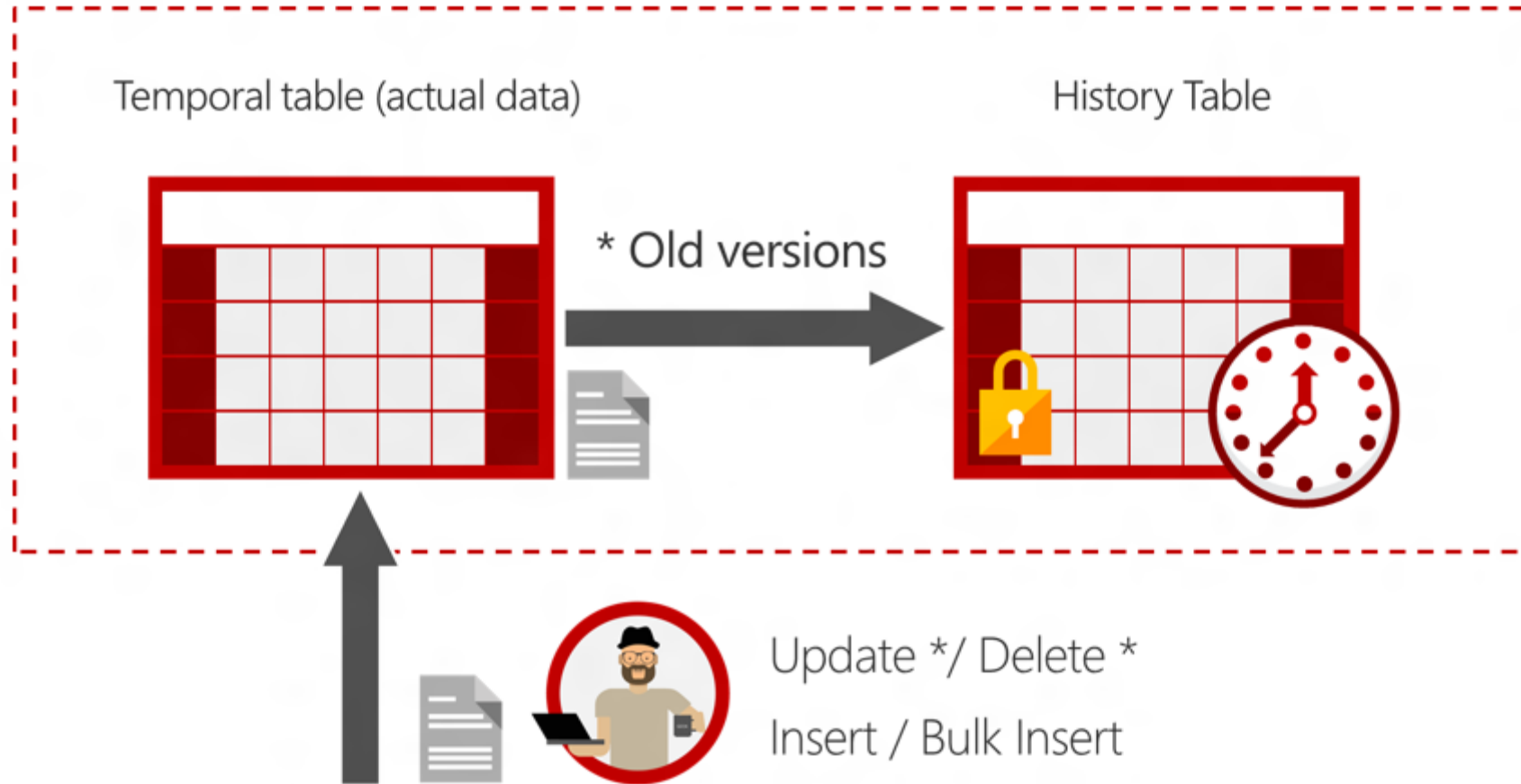
Available in SQL Server 2016, SQL Azure

Every time a row is changed, SQL Server keeps a snapshot of the old row values

SQL constructs allow us to see what the value was at any given date/time

No need to implement your own triggers, history tables and views

# Temporal Table Concept



# CREATE TABLE Syntax

```
CREATE TABLE Contacts
(
    ContactID          INT IDENTITY(1,1)          NOT NULL,
    FirstName          VARCHAR(30)                NOT NULL,
    LastName           VARCHAR(30)                NOT NULL,
    CompanyName        VARCHAR(30)                NULL,
    PhoneNumber        VARCHAR(20)                NULL,
    Email              VARCHAR(50)                NULL,
    ValidFrom          DATETIME2(3) GENERATED ALWAYS AS ROW START,
    ValidTo            DATETIME2(3) GENERATED ALWAYS AS ROW END,
    PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo),
    CONSTRAINT PK_Contacts PRIMARY KEY (ContactId)
)
WITH (SYSTEM_VERSIONING = ON
      (HISTORY_TABLE = dbo.ContactsHistory));
```

# Hiding ValidFrom and ValidTo Columns

```
-- Hide the ValidFrom and ValidTo columns
```

```
ALTER TABLE Contacts
```

```
    ALTER COLUMN ValidFrom ADD HIDDEN;
```

```
ALTER TABLE Contacts
```

```
    ALTER COLUMN ValidTo ADD HIDDEN;
```

```
-- Show the ValidFrom and ValidTo Columns
```

```
ALTER TABLE Contacts
```

```
    ALTER COLUMN ValidFrom DROP HIDDEN;
```

```
ALTER TABLE Contacts
```

```
    ALTER COLUMN ValidTo DROP HIDDEN;
```



# Querying Temporal Tables

```
-- Get all currently valid rows (query just like a normal table)
```

```
SELECT *  
  FROM Contacts;
```

```
-- Get the rows that were valid at a certain date/time
```

```
SELECT *  
  FROM Contacts  
    FOR SYSTEM_TIME AS OF '2017-04-12 14:00:00.000';
```

```
-- Get all versions of all rows that have ever existed in the table
```

```
SELECT *  
  FROM Contacts  
    FOR SYSTEM_TIME ALL;
```

```
-- Get all versions of a row for a given key in a table (basically a history of this row)
```

```
SELECT *  
  FROM Contacts  
    FOR SYSTEM_TIME ALL  
 WHERE ContactId = 1;
```

# Temporal Table Query Qualifiers

Expression	Description
AS OF <date time>	Gets all of the rows that were active at the specified date and time
FROM <start time> TO <end time>	Gets all of the rows that were active at any point during the specified time period, excluding rows that were only active at the start time or end time
BETWEEN <start time> AND <end time>	Gets all rows active in the time period including rows that became active at the ending time of the period
CONTAINED IN (<start time>, <end time>)	Gets all rows that were opened and closed in the period specified
ALL	Get all rows in the table and the history table

# Converting an Existing Table to a Temporal Table

```
-- Step 1 - Add the columns and period
ALTER TABLE Contacts ADD
    ValidFrom DATETIME2(3) GENERATED ALWAYS AS ROW START
        NOT NULL DEFAULT '1900-01-01 00:00:00.000',
    ValidTo    DATETIME2(3) GENERATED ALWAYS AS ROW END
        NOT NULL DEFAULT '9999-12-31 23:59:59.999',
    PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo);

-- Step 2 - Turn on System Versioning and define the history table
ALTER TABLE Employees
    SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.ContactsHistory));
```

# JSON Support

Available in SQL Server 2016, SQL Azure

JSON data stored in a VARCHAR data type

Functions available to interact with JSON data

Allows us to mix relational and NoSQL data models in the same database

# Storing JSON in Tables

```
CREATE TABLE WeatherDataJson
(
    ObservationId      INT IDENTITY(1,1) NOT NULL,
    StationCode        VARCHAR(10)      NOT NULL,
    City               VARCHAR(30)       NOT NULL,
    State              VARCHAR(2)        NOT NULL,
    ObservationDate     DATETIME          NOT NULL,
    ObservationData     VARCHAR(4000)     NOT NULL,
    CONSTRAINT PK_WeatherDataJson
        PRIMARY KEY (ObservationId)
);
```

# Getting Data From a JSON Column

JSON_VALUE	Extract a scalar value from a JSON String
JSON_QUERY	Extract an object or an array from a JSON String
OPENJSON	Table value function used for parsing JSON and returning a rowset view of the data

# Querying Scalar Values

```
SELECT
    ObservationId,
    StationCode,
    City,
    State,
    ObservationDate,
    JSON_VALUE(ObservationData, '$.dryBulbFarenheit') As Temperature,
    JSON_VALUE(ObservationData, '$.relativeHumidity') As Humidity,
    ObservationData
FROM WeatherDataJson
WHERE State = 'WI'
    AND City = 'Milwaukee'
    AND ObservationDate > '2016-08-15'
    AND ObservationDate < '2016-08-16';
```

## Exposing JSON Values With Computed Columns

```
-- Create a computed column on the table
ALTER TABLE WeatherDataJson
  ADD Temperature AS
    (JSON_VALUE(ObservationData, '$.dryBulbFarenheit'));

--This column can now be queried like any other column
```



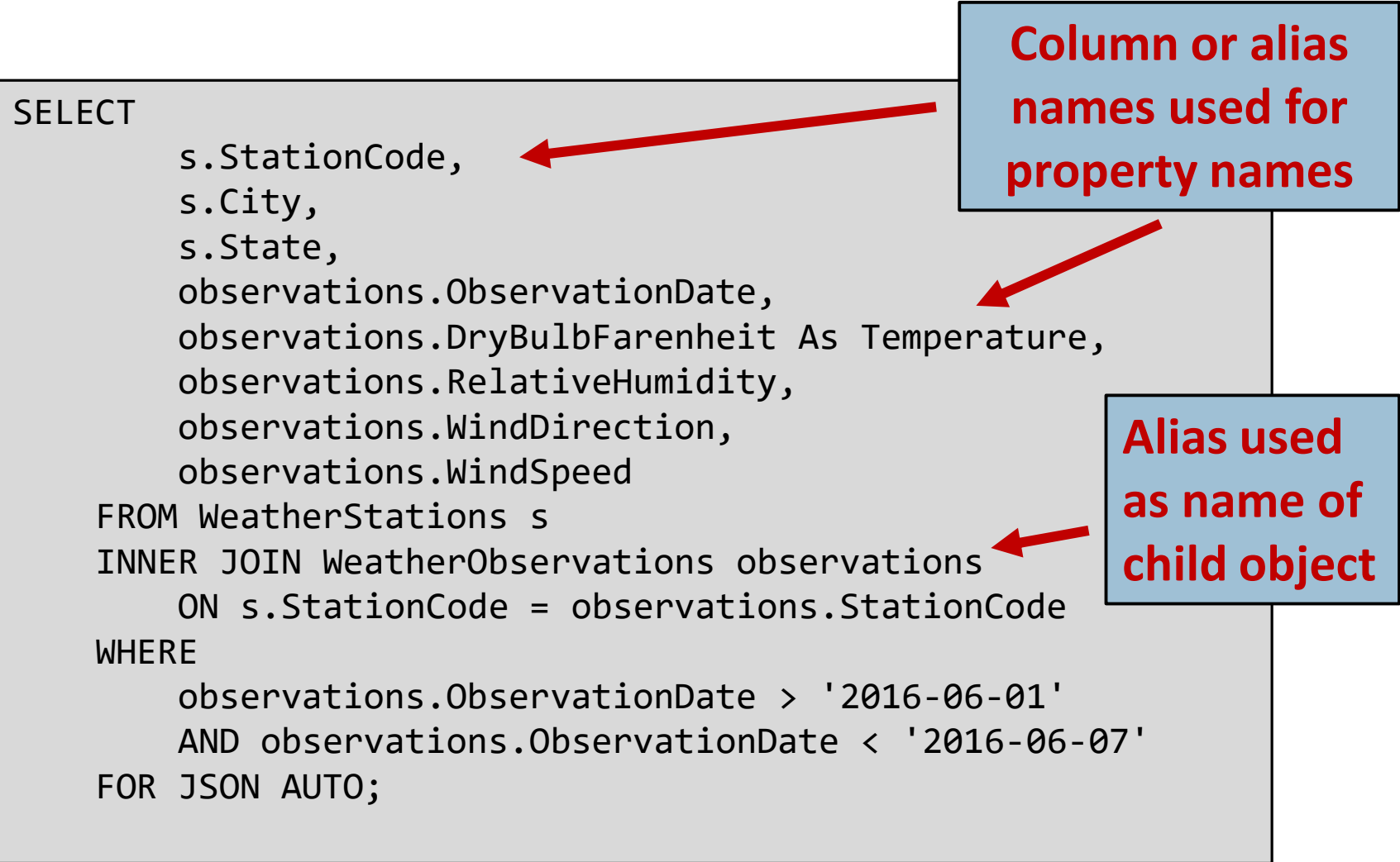
# Querying Values in JSON Arrays

```
SELECT
    ObservationId,
    StationCode,
    City,
    State,
    ObservationDate,
    CONVERT(datetime, JSON_VALUE(Observations.[Value], '$.observationDateTime'), 126)
As ObservationTime,
    JSON_VALUE(Observations.[Value], '$.dryBulbFarenheit') As Temperature,
    JSON_VALUE(Observations.[Value], '$.relativeHumidity') As Humidity,
    JSON_VALUE(Observations.[Value], '$.windDirection') As WindDirection,
    JSON_VALUE(Observations.[Value], '$.windSpeed') As WindSpeed
FROM DailyWeatherDataJson d
CROSS APPLY OPENJSON(JSON_QUERY(ObservationData, '$.weatherObservations'))
Observations
WHERE City = 'Milwaukee'
AND State = 'WI'
AND ObservationDate = ('2016-08-15');
```

# Return Results As JSON

```
SELECT
    s.StationCode,
    s.City,
    s.State,
    observations.ObservationDate,
    observations.DryBulbFarenheit As Temperature,
    observations.RelativeHumidity,
    observations.WindDirection,
    observations.WindSpeed
FROM WeatherStations s
INNER JOIN WeatherObservations observations
    ON s.StationCode = observations.StationCode
WHERE
    observations.ObservationDate > '2016-06-01'
    AND observations.ObservationDate < '2016-06-07'
FOR JSON AUTO;
```

**Column or alias  
names used for  
property names**



**Alias used  
as name of  
child object**

# Window Functions

Available in SQL Server 2008 and later, SQL Azure

Extended functionality over traditional GROUP BY queries

Can partition and aggregate data by different criteria in the same statement

Greatly enhances reporting capabilities in SQL Server

# Window Functions

## Ranking Functions

---

ROW\_NUMBER  
RANK  
DENSE\_RANK  
NTILE

## Aggregate Functions

---

AVG  
COUNT  
SUM  
MIN  
MAX  
STDEV  
STDEVP  
VAR  
VARP  
CHECKSUM\_AGG

## Analytic Functions

---

LAG  
LEAD  
FIRST\_VALUE  
LAST\_VALUE  
CUME\_DIST  
PERCENTILE\_CONT  
PERCENTILE\_DISC  
PERCENTILE\_RANK

# Window Function Syntax

```
SELECT DISTINCT
    City,
    State
    CONVERT (DATE, ObservationDate) AS SummaryDate,
    RANK ()
        OVER (PARTITION BY City, State, CONVERT (DATE, ObservationDate)
            ORDER BY DailyHighTemp DESC) As HighTemp
FROM DailyWeatherSummaries o
WHERE
    State = 'WI'
    AND ObservationDate BETWEEN '2016-05-01' AND '2016-06-01';
```

- |                     |   |
|---------------------|---|
| <b>PARTITION BY</b> | Splits the result set into different partitions or groups using these columns                                 |
| <b>ORDER BY</b>     | Orders the rows within each partition for functions where order is important (not used in for some functions) |

# Window Function Grouping

**RANK()**

**OVER (PARTITION BY City, State, CONVERT (DATE, ObservationDate)  
ORDER BY DailyHighTemp DESC) As HighTemp**

City	State	SummaryDate	DailyHighTemp	Rank()	
Chicago	IL	2017-06-01	65	1	Partition Group
Chicago	IL	2017-06-01	63	2	
Chicago	IL	2017-06-01	62	3	
Chicago	IL	2017-06-02	67	1	Partition Group
Chicago	IL	2017-06-02	65	2	
Chicago	IL	2017-06-02	63	3	
Milwaukee	WI	2017-06-01	71	1	Partition Group
Milwaukee	WI	2017-06-01	70	2	
Milwaukee	WI	2017-06-01	68	3	

Partition Columns

Sort Column

# Ranking Window Functions

ROW_NUMBER()	Creates sequential number of a row within a partition of a result set, starting at 1 for the first row in each partition
RANK()	Returns the rank of each row within the partition of a result set. If there two or more values tie for a rank, then there will be a gap to the next ranking
DENSE_RANK()	Returns the rank of each row within the partition of a result set without any gaps between rankings in event of multiple values for the same rank
NTILE(<integer>)	Distributes the rows in the partition into the specified number of buckets (groups) and shows what bucket that row falls in.

# Analytic Window Functions

<code>FIRST_VALUE(&lt;column&gt;)</code>	Returns the first value of the partition as sorted by the ORDER BY clause
<code>LAST_VALUE(&lt;column&gt;)</code>	Returns the last value of the partition as sorted by the ORDER BY clause
<code>LAG(&lt;column&gt;,     &lt;offset&gt;, &lt;default&gt;)</code>	Returns the preceding value in the partition. Offset (optional) allows you to reach back multiple rows. Default (optional) allows a default value to be specified
<code>LEAD(&lt;column&gt;,       &lt;offset&gt;, &lt;default&gt;)</code>	Returns the next value in the partition. Offset (optional) allows you to reach forward multiple rows. Default (optional) allows a default value to be specified



# MERGE Statement

Available in SQL Server 2008 and later, SQL Azure

Essentially provides and “UPSERT” capability

Useful when doing bulk updates to tables

Can be useful for setting reference data in DEV databases

# Problem Scenario

**Source Table**



\*May contain records  
already in our data table



**Target Table**



Use a MERGE statement  
to perform an UPSERT

# Resources

## Temporal Tables

<https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables>

## JSON Functions

<https://docs.microsoft.com/en-us/sql/t-sql/functions/json-functions-transact-sql>

<https://www.simple-talk.com/sql/learn-sql-server/json-support-in-sql-server-2016/>

## Window Functions

<https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql>

<https://www.simple-talk.com/sql/learn-sql-server/window-functions-in-sql-server/>