Time series data T-SQL functions

Last updated by | Peter Hewitt | Nov 14, 2022 at 6:43 AM PST

Contents

Introduction

When dealing with time series data, it's often possible that the time series data has missing values for the attributes. It's also possible that, because of the nature of the data, or because of interruptions in data collection, there are time gaps in the dataset.

For example, when collecting energy usage statistics for a smart device, whenever the device isn't operational there will be gaps in the usage statistics. Similarly, in a machine telemetry data collection scenario, it's possible that the different sensors are configured to emit data at different frequencies, resulting in missing values for the sensors. For example, if there are two sensors, voltage and pressure, configured at 100 Hz and 10-Hz frequency respectively, the voltage sensor will emit data every one-hundredth of a second, while the pressure sensor will only emit data every one-tenth of a second.

The following table describes a MachineTelemetry dataset, which was collected at a one-second interval.

```
CREATE TABLE MachineTelemetry
([timestamp] datetime,
 VoltageReading decimal(18,6),
 PressureReading decimal(18,6))
 GO

INSERT INTO MachineTelemetry ([timestamp],VoltageReading,PressureReading)
values
('2020-09-07 06:14:41.000',164.990400,97.223600),
('2020-09-07 06:14:42.000',162.241300,93.992800),
('2020-09-07 06:14:43.000',163.271200,NULL),
('2020-09-07 06:14:44.000',161.368100,93.403700),
('2020-09-07 06:14:45.000',NULL,NULL),
('2020-09-07 06:14:46.000',NULL,98.364800),
('2020-09-07 06:14:49.000',NULL,94.098300),
('2020-09-07 06:14:51.000',157.695700,103.359100),
('2020-09-07 06:14:52.000',157.019200,NULL),
('2020-09-07 06:14:54.000',NULL,95.352000),
('2020-09-07 06:14:56.000',159.183500,100.748200)
```

The output from the MachineTelemetry table:

```
timestamp               VoltageReading  PressureReading
----------------------- --------------- ----------------
2020-09-07 06:14:41.000 164.990400      97.223600
2020-09-07 06:14:42.000 162.241300      93.992800
2020-09-07 06:14:43.000 163.271200      NULL
2020-09-07 06:14:44.000 161.368100      93.403700
2020-09-07 06:14:45.000 NULL            NULL
2020-09-07 06:14:46.000 NULL            98.364800
2020-09-07 06:14:49.000 NULL            94.098300
2020-09-07 06:14:51.000 157.695700      103.359100
2020-09-07 06:14:52.000 157.019200      NULL
2020-09-07 06:14:54.000 NULL            95.352000
2020-09-07 06:14:56.000 159.183500      100.748200
```

There are two important characteristics of the preceding dataset.

- The dataset doesn't contain any data points related to several timestamps `2020-09-07 06:14:47.000`, `2020-09-07 06:14:48.000`, `2020-09-07 06:14:50.000`, `2020-09-07 06:14:53.000`, and `2020-09-07 06:14:55.000`. These timestamps are *gaps* in the dataset.
- There are missing values, represented as `null`, for the Voltage and pressure readings.

Gap filling

Gap filling is a technique that helps create contiguous, ordered set of timestamps to ease the analysis of time series data. The easiest way to fill gaps in the time series dataset is to define a temporary table with the desired time distribution and then do a `LEFT OUTER JOIN` or a `RIGHT OUTER JOIN` operation on the dataset table.

Taking the MachineTelemetry data as an example, the following query can be used to generate contiguous, ordered set of timestamps for analysis.

Note: The query below generates the missing rows, with the timestamp values and `null` values for the attributes.

```
CREATE TABLE #SeriesGenerate(dt datetime Primary key Clustered)
GO

DECLARE @startdate datetime = '2020-09-07 06:14:41.000', @endtime datetime = '2020-09-07 06:14:56.000'
WHILE (@startdate <= @endtime)
BEGIN
  INSERT into #SeriesGenerate values (@startdate)
   SET @startdate = DATEADD(SECOND, 1, @startdate)
END

SELECT a.dt as timestamp, b.VoltageReading, b.PressureReading
FROM #SeriesGenerate a
  LEFT OUTER JOIN MachineTelemetry b ON a.dt = b.[timestamp]
```

The output now contains all *one-second* timestamps in the specified range:

```
timestamp               VoltageReading    PressureReading
----------------------- ----------------- ----------------
2020-09-07 06:14:41.000 164.990400        97.223600
2020-09-07 06:14:42.000 162.241300        93.992800
2020-09-07 06:14:43.000 163.271200        NULL
2020-09-07 06:14:44.000 161.368100        93.403700
2020-09-07 06:14:45.000 NULL              NULL
2020-09-07 06:14:46.000 NULL              98.364800
2020-09-07 06:14:47.000 NULL              NULL
2020-09-07 06:14:48.000 NULL              NULL
2020-09-07 06:14:49.000 NULL              94.098300
2020-09-07 06:14:50.000 NULL              NULL
2020-09-07 06:14:51.000 157.695700        103.359100
2020-09-07 06:14:52.000 157.019200        NULL
2020-09-07 06:14:53.000 NULL              NULL
2020-09-07 06:14:54.000 NULL              95.352000
2020-09-07 06:14:55.000 NULL              NULL
2020-09-07 06:14:56.000 159.183500        100.748200
```

New T-SQL syntax added

The preceding query generated the missing timestamps for data analysis, however it did not replace any of the missing values (represented as null) for `voltage` and `pressure` readings.

In 2022, for SQL Database and SQL Managed Instance new syntax was added to the T-SQL `LAST_VALUE()` and `FIRST_VALUE()` functions, which provide mechanisms to impute missing values, based on the preceding or following values in the dataset. The new syntax adds `IGNORE NULLS` and `RESPECT NULLS` clause to the `LAST_VALUE()` and `FIRST_VALUE()` functions.

In addition, two new functions `DATE_BUCKET()` and `GENERATE_SERIES()` were added. The `DATE_BUCKET()` function returns the date-time value corresponding to the start of each date-time bucket and the `GENERATE_SERIES()` function generates a series of numbers within a given interval. The interval and the step between series values are defined by the user.

> *Note: GENERATE_SERIES requires the compatibility level to be at least 160. When the compatibility level is less than 160, SQL Server is unable to find the GENERATE_SERIES function.*

Example 1: This query on the MachineTelemetry dataset computes the missing values using the `LAST_VALUE()` function, where missing values are replaced with the last observed value in the dataset.

```
SELECT
    timestamp,
    VoltageReading As OriginalVoltageValues,
    LAST_VALUE(VoltageReading) IGNORE NULLS OVER (ORDER BY timestamp) As ImputedUsingLastValue,
    PressureReading As OriginalPressureValues,
    LAST_VALUE(PressureReading) IGNORE NULLS OVER (ORDER BY timestamp) As ImputedUsingLastValue
FROM MachineTelemetry
ORDER BY timestamp
```

```
timestamp               OrigVoltageVals  ImputedVoltage OrigPressureVals  ImputedPressure
----------------------- ---------------- -------------- ----------------- ---------------
2020-09-07 06:14:41.000 164.990400       164.990400     97.223600         97.223600
2020-09-07 06:14:42.000 162.241300       162.241300     93.992800         93.992800
2020-09-07 06:14:43.000 163.271200       163.271200     NULL              93.992800
2020-09-07 06:14:44.000 161.368100       161.368100     93.403700         93.403700
2020-09-07 06:14:45.000 NULL             161.368100     NULL              93.403700
2020-09-07 06:14:46.000 NULL             161.368100     98.364800         98.364800
2020-09-07 06:14:49.000 NULL             161.368100     94.098300         94.098300
2020-09-07 06:14:51.000 157.695700       157.695700     103.359100        103.359100
2020-09-07 06:14:52.000 157.019200       157.019200     NULL              103.359100
2020-09-07 06:14:54.000 NULL             157.019200     95.352000         95.352000
2020-09-07 06:14:56.000 159.183500       159.183500     100.748200        100.748200
```

Example 2: This query imputes the missing values using both the `LAST_VALUE()` and the `FIRST_VALUE()` functions. For, the output column `ImputedVoltage` the missing values are replaced by the last observed value, while for the output column `ImputedPressure` the missing values are replaced by the next observed value in the dataset.

```
SELECT
  dt AS timestamp, VoltageReading As OrigVoltageVals,
  LAST_VALUE(VoltageReading) IGNORE NULLS OVER (ORDER BY dt) As ImputedVoltage, PressureReading As OrigPressur
  FIRST_VALUE(PressureReading) IGNORE NULLS OVER (ORDER BY dt ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
FROM
(SELECT a.dt, b.VoltageReading,b.PressureReading
 FROM #SeriesGenerate a
   LEFT OUTER JOIN MachineTelemetry b ON a.dt = b.[timestamp]) A
ORDER BY timestamp
```

```
timestamp               OrigVoltageVals  ImputedVoltage  OrigPressureVals  ImputedPressure
----------------------- ---------------- --------------- ----------------- ---------------
2020-09-07 06:14:41.000 164.990400       164.990400      97.223600         97.223600
2020-09-07 06:14:42.000 162.241300       162.241300      93.992800         93.992800
2020-09-07 06:14:43.000 163.271200       163.271200      NULL              93.403700
2020-09-07 06:14:44.000 161.368100       161.368100      93.403700         93.403700
2020-09-07 06:14:45.000 NULL             161.368100      NULL              98.364800
2020-09-07 06:14:46.000 NULL             161.368100      98.364800         98.364800
2020-09-07 06:14:47.000 NULL             161.368100      NULL              94.098300
2020-09-07 06:14:48.000 NULL             161.368100      NULL              94.098300
2020-09-07 06:14:49.000 NULL             161.368100      94.098300         94.098300
2020-09-07 06:14:50.000 NULL             161.368100      NULL              103.359100
2020-09-07 06:14:51.000 157.695700       157.695700      103.359100        103.359100
2020-09-07 06:14:52.000 157.019200       157.019200      NULL              95.352000
2020-09-07 06:14:53.000 NULL             157.019200      NULL              95.352000
2020-09-07 06:14:54.000 NULL             157.019200      95.352000         95.352000
2020-09-07 06:14:55.000 NULL             157.019200      NULL              100.748200
2020-09-07 06:14:56.000 159.183500       159.183500      100.748200        100.748200
```

Note: The above query uses the `FIRST_VALUE()` function to replace missing values with the next observed value. The same result can be achieved by using the `LAST_VALUE()` function with a `ORDER BY <ordering_column> DESC` clause.

Example 3: Calculate `DATE_BUCKET` with a bucket width of 1 from the origin time. Each of these statements increments `DATE_BUCKET` with a bucket width of 1 from the origin time (default origin value = 1900-01-01 00:00:00.000):

```
DECLARE @date DATETIME2 = '2020-04-30 21:21:21';
SELECT 'Week', DATE_BUCKET(WEEK, 1, @date)
UNION ALL
SELECT 'Day', DATE_BUCKET(DAY, 1, @date)
UNION ALL
SELECT 'Hour', DATE_BUCKET(HOUR, 1, @date)
UNION ALL
SELECT 'Minutes', DATE_BUCKET(MINUTE, 1, @date)
UNION ALL
SELECT 'Seconds', DATE_BUCKET(SECOND, 1, @date);
```

```
Week    2020-04-27 00:00:00.0000000
Day     2020-04-30 00:00:00.0000000
Hour    2020-04-30 21:00:00.0000000
Minutes 2020-04-30 21:21:00.0000000
Seconds 2020-04-30 21:21:21.0000000
```

Example 4: Generate a series of decimal values between 0.0 and 1.0 in increments of 0.1.

```
DECLARE @start decimal(2, 1) = 0.0;
DECLARE @stop decimal(2, 1) = 1.0;
DECLARE @step decimal(2, 1) = 0.1;

SELECT value
FROM GENERATE_SERIES(@start, @stop, @step);
```

```
value
-----
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
```

Internal Reference

Internal video demonstrating the new capabilities for Time Series functions
GENERATE_SERIES function detailed error messages

Public Doc Reference

Date and time data types and functions (Transact-SQL)
DATE_BUCKET (Transact-SQL)
GENERATE_SERIES (Transact-SQL)
FIRST_VALUE (Transact-SQL)
LAST_VALUE (Transact-SQL)

Root Cause Classification

Cases resolved by this TSG should be coded to the following root cause:
SQL Database/Performance and Query Execution

How good have you found this content?