Basic aggregate functions

TIME SERIES ANALYSIS IN SQL SERVER

Kevin Feasel
CTO, Envizage

Key aggregation functions

Counts

`COUNT()`

`COUNT_BIG()`

`COUNT(DISTINCT)`

Other Aggregates

`SUM()`

`MIN()`

`MAX()`

What counts with COUNT()

Number of Rows

```
COUNT(*)
```

```
COUNT(1)
```

```
COUNT(1/0)
```

Non-NULL Values

```
COUNT(d.YR)
```

```
COUNT(NULLIF(d.YR, 1990))
```

Distinct counts

```
SELECT
    COUNT(DISTINCT c.CalendarYear) AS Years,
    COUNT(DISTINCT NULLIF(c.CalendarYear, 2010)) AS Y2
FROM dbo.Calendar c;
```

the distinct number of non-2010 years

Years	Y2	
50	49	

Filtering aggregates with CASE

```
SELECT
    MAX(CASE WHEN ir.IncidentTypeID = 1
             THEN ir.IncidentDate
             ELSE NULL
        END) AS I1,
    MAX(CASE WHEN ir.IncidentTypeID = 2
             THEN ir.IncidentDate
             ELSE NULL
        END) AS I2,
FROM dbo.IncidentRollup ir;
```

returns the incident date when we match incident types and returns NULL otherwise

I1	I2	
2020-06-30	2020-06-29	

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER

Statistical aggregate functions

TIME SERIES ANALYSIS IN SQL SERVER

Kevin Feasel
CTO, Envizage

Statistical aggregate functions

`AVG()`	Mean
`STDEV()`	Standard Deviation
`STDEVP()`	Population Standard Deviation
`VAR()`	Variance
`VARP()`	Population Variance

What about median?

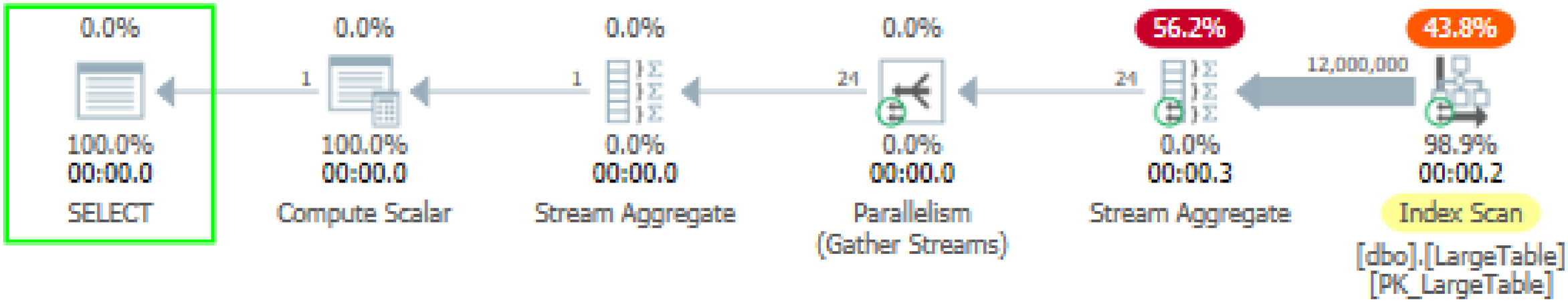
```
SELECT TOP(1)
    PERCENTILE_CONT(0.5)
        WITHIN GROUP (ORDER BY l.SomeVal DESC)
        OVER () AS MedianIncidents
FROM dbo.LargeTable l;
```

a window function, it will return one row for every row sent in

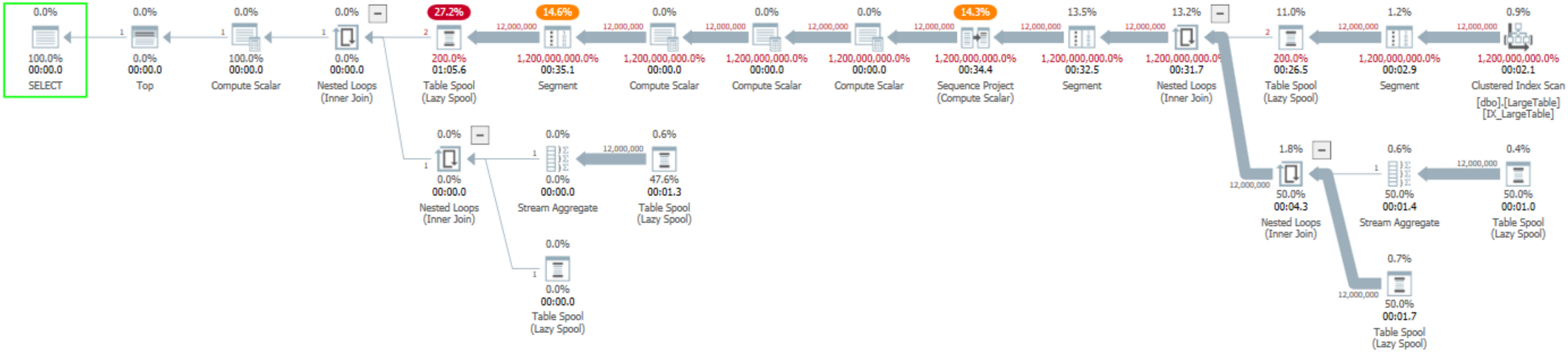
how you order the data set

allow us to partition the data

But how bad is it?



This bad


0.0%
100.0%
00:00.0
SELECT
0.0%
0.0%
00:00.0
Top
0.0%
100.0%
00:00.0
Compute Scalar
0.0%
0.0%
00:00.0
Nested Loops
(Inner Join)
27.2%
200.0%
01:05.6
Table Spool
(Lazy Spool)
14.6%
1,200,000,000.0%
00:35.1
Segment
0.0%
1,200,000,000.0%
00:00.0
Compute Scalar
0.0%
1,200,000,000.0%
00:00.0
Compute Scalar
0.0%
1,200,000,000.0%
00:00.0
Compute Scalar
14.3%
1,200,000,000.0%
00:34.4
Sequence Project
(Compute Scalar)
13.5%
1,200,000,000.0%
00:32.5
Segment
13.2%
1,200,000,000.0%
00:31.7
Nested Loops
(Inner Join)
11.0%
200.0%
00:26.5
Table Spool
(Lazy Spool)
1.2%
1,200,000,000.0%
00:02.9
Segment
0.9%
1,200,000,000.0%
00:02.1
Clustered Index Scan
[dbo].[LargeTable]
[IX_LargeTable]
0.0%
0.0%
00:00.0
Nested Loops
(Inner Join)
0.0%
0.0%
00:00.0
Stream Aggregate
0.6%
47.6%
00:01.3
Table Spool
(Lazy Spool)
0.0%
0.0%
00:00.0
Table Spool
(Lazy Spool)
1.8%
50.0%
00:04.3
Nested Loops
(Inner Join)
0.6%
50.0%
00:01.4
Stream Aggregate
0.4%
50.0%
00:01.0
Table Spool
(Lazy Spool)
0.7%
50.0%
00:01.7
Table Spool
(Lazy Spool)

The cost of median

	Median	Mean
Est. Cost	95.7%	4.3%
Duration	68.5s	0.37s
CPU	68.5s	8.1s
Reads	72,560,946	39,468
Writes	87,982	0

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER

Downsampling and upsampling data

TIME SERIES ANALYSIS IN SQL SERVER

Kevin Feasel
CTO, Envizage

Data in nature

```
SELECT
    SomeDate
FROM dbo.SomeTable
```

SomeDate
2019-08-11 06:14:29.990
2019-08-11 11:07:37.633
2019-08-11 14:08:00.337

Downsampling data

```
SELECT
    CAST(SomeDate AS DATE) AS SomeDate
FROM dbo.SomeTable
```

SomeDate
2019-08-11
2019-08-11
2019-08-11

Further downsampling

```
SELECT
    DATEADD(HOUR, DATEDIFF(HOUR, 0, SomeDate), 0) AS SomeDate
FROM dbo.SomeTable
```

SomeDate
2019-08-11 06:00:00.000
2019-08-11 11:00:00.000
2019-08-11 14:00:00.000

We figure out the number of hours from SQL Server's starting point (time 0) until our customer visit start. The 'DATEDIFF' function loops off any unused date or time parts, so anything lower than the hour goes away and 'DATEDIFF()' returns an integer representing the number of hours from time 0 until 'SomeDate'.
Then, we add the number of hours to time 0, giving us a rounded total.
The end result is a 'DATETIME' data type rounded to the nearest hour.

What about upsampling?

Downsampling

- Aggregate data
- Can usually sum or count results
- Provides a higher-level picture of the data
- Acceptable for most purposes

Upsampling

- Disaggregate data
- Need an allocation rule
- Provides artificial granularity
- Acceptable for data generation, calculated averages

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER

Grouping by ROLLUP, CUBE, and GROUPING SETS

TIME SERIES ANALYSIS IN SQL SERVER

Kevin Feasel
CTO, Envizage

Hierarchical rollups with ROLLUP

```
SELECT
    t.Month,
    t.Day,
    SUM(t.Events) AS Events
FROM Table
GROUP BY
    t.Month,
    t.Day
WITH ROLLUP
ORDER BY
    t.Month,
    t.Day;
```

The ‘WITH ROLLUP’ clause comes after ‘GROUP BY’ and tells SQL Server to roll up the data.

Month	Day	Events
NULL	NULL	100
1	NULL	60
1	1	3
1	2	4
…	…	…
2	NULL	40
2	1	8

‘ROLLUP’ will take each combination of the first column (month), followed by each matching value in the second column, and so on, showing our aggregates for each.

Cartesian aggregation with CUBE

```
SELECT
    t.IncidentType,
    t.Office,
    SUM(t.Events) AS Events
FROM Table
GROUP BY
    t.IncidentType,
    t.Office
WITH CUBE
ORDER BY
    t.IncidentType,
    t.Office;
```

For cases where you want to see the full combination of all aggregations between columns, CUBE is at our disposal.

IncidentType	Office	Events
NULL	NULL	250
NULL	NY	70
NULL	CT	180
T1	NULL	55
T1	NY	30
T1	CT	25

Define grouping sets with GROUPING SETS

```
SELECT
    t.IncidentType,
    t.Office,
    SUM(t.Events) AS Events
FROM Table
GROUP BY GROUPING SETS
(
  (t.IncidentType, t.Office),
  ()
)
ORDER BY
    t.IncidentType,
    t.Office;
```

This results in one row with the grant total followed by each of the specific combinations of incident type and office.
If we then want to include separate aggregates like all of the incident types broken out regardless of office, we can add those as additional grouping sets.

IncidentType	Office	Events
NULL	NULL	250
T1	NY	30
T1	CT	25
T2	NY	10
T2	CT	110
T3	NY	30
T3	CT	45

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER