Teaching Case

An Instructor's Tutorial and Student Project for Extending SQL Implementation to Calculate Percentile Aggregates for Ungrouped Data

John N. Dyer
jdyer@GeorgiaSouthern.Edu
Department of Enterprise Systems and Analytics
Georgia Southern University
Statesboro, GA 30458

Abstract

More and more organizations are becoming data-driven, and more so than ever, teachers, students, and practitioners are expected to possess an adequate level of skills in basic Structured Query Language (SQL) as well as advanced skills. It is increasingly important that all those practicing SQL possess a higher level of SQL proficiency than a basic database class. Although relational database management systems (RDBMS) encompass many query tools, they were not designed to behave like spreadsheets or analytical software. As such, one often goes back and forth between the database and software or spreadsheet applications to organize and summarize data before analysis. While the RDBMS allow one to effectively organize data, spreadsheets enable one to more easily summarize data using basic aggregate descriptive statistics such as the mean, median, mode, percentiles, counts, frequency distributions, etc. Most RDBMS allow only a small set of standard SQL aggregate functions such as for calculating the mean, sums, counts, variances, and standard deviations. This teaching tutorial tutorial presents a set of queries that allow higher-level learning of aggregate queries within the RDBMS, including percentiles, deciles, and quartiles, as well as the median for describing ungrouped data. In doing so, lesser-studied SQL aggregates and functions are introduced including the MODULO operator, the IIF comparison operator, and several formatting functions. It is hoped that this tutorial can serve as a resource to facilitate higher learning skills and expand the capabilities of teachers, students, and practitioners in SQL. The complete Access database including tables and queries is available from the author upon request.

Keywords: Database System, SQL, Relational Algebra, Aggregate Function, Percentile.

1. INTRODUCTION

The purpose of this teaching tutorial is primarily to be a teaching case type application to enhance and expand the SQL skill sets for teachers, students, and everyday users of SQL in academics and industry. The topics in this tutorial have experienced little formal addressing in the literature or in practice. Most examples in this tutorial use small data sets for the sake of visualization but can readily be applied to very large data sets, limited only by the user's computational power. It also introduces an expanded skill set beyond the basic SELECT statement, by allowing readers to see the breadth of possibilities for calculating aggregates, functions and expressions related to database attributes. This tutorial is meant to be timely in terms of bridging a gap between classroom SQL and within-field practice, wherein most IS/CS and analytics persons are expected to "hit-the-ground-running" in SQL. By digesting SQL topics at a higher level, it is hoped that the reader can enhance their SQL skill set in terms of thinking of how much more can be achieved beyond basic

SQL when one knows the limits and opportunities within the practice of SQL programming.

SQL has been primarily the domain of information systems, database and computer science educators, students, and practitioners. As the world is progressing towards digitalization, there has been a shift wherein many organizations are now data-driven and depend heavily on database and data analytics practitioners possessing SQL as a top skill. As such, SQL has become ubiquitous across most functional areas of modern organizations and has become imperative that a greater cross-section of teachers, students and practitioners possess more than just basic SQL skills, hence, the overreaching purpose of this tutorial. It is hoped that this tutorial will make a meaningful contribution to higher-level SQL skills in the classroom, as well as in practice.

2. BASIC SQL AND AGGREGATE FUNCTIONS

SQL readily allows the organizing and summarizing data within the domain of the SELECT and JOIN operators, projecting various. aggregate functions of attributes as output, calculated fields (expressions), arithmetic and comparison operators, IF statements, and formatting, among others. A good SQL tutorial can be viewed online at *SQL operators* (n.d). SQL includes a small set of built-in aggregate functions including the AVEREGE, COUNT, SUM, MINIMUM, MAXIMUM, VARIENCE, STANDARD DEVIATION, and the FIRST, LAST and TOP aggregates. Standard SQL was not designed to calculate aggregates such as the percentiles, deciles, quintiles, quartiles, median, cumulative frequency distributions, mode, etc. For these undefined SQL aggregates, one might asynchronously export data into an external software application such as an Excel spreadsheet. Some practitioners may find this acceptable, not having the need to expand the set of aggregate functions, but if the RDBMS is often used to provide real-time data aggregation, such as used in managerial dashboards and visual summaries, the external software approach may not always the best approach. It works best when the external software has a synchronous link to the RDBMS.

Aggregates discussed in this tutorial are often important in providing a complete organization and summary of large data sets that may be used on a case-by-case basis with other descriptive aggregates, or as additional metrics in a larger summary like a managerial dashboard.

3. PERCENTILE AGGREGATES

What standard RDBMS SQL leaves out are aggregates used to summarize data using percentile aggregates, like deciles, quartiles, quintiles, and the median, as well as frequency and percent frequency distributions of grouped numerical and categorial data. All spreadsheet and analytical software applications provide numerous aggregate functions that one would use to summarize a dataset.

Percentiles are relative locations of data in an ordered data set. The *P*th percentile value (*p*) of a set of data is the value at which *p* percent of the data is below it. Along with other numerical summaries used in statistics and data analytics such as the count, average, variance, and standard deviation, percentiles provide other measures of the distribution of data values. Common percentiles include deciles (10th p, 20th p, … 90th p), quintiles (20th p, 40th p, 60th p, 80th p) and quartiles (25th p, 50th p and 75th p).

Percentiles are largely used in statistics and analytics, and in the everyday life of data consumers. Percentiles are used to describe the distribution of values such as test scores, health indicators, and other measurements. That is, given a percentile value, we calculate the value's relative standing in the data set, even if we do not know the actual data value. Likewise, analysts use the interquartile range (IQR), calculated as Q3-Q2, as a measure of variability (dispersion) for the middle 50 percent of data, and use the IQR as a tool to detect outliers in data. The minimum is also known as the 0th *p*, while the maximum is the 100th *p*, and the median is the 50th *p*. If one's test score is at the 75th percentile, that tells us that 75% of the other scores fall below that score.

In general, the percentile value of a data value is given by Percentile = [(number of values below a selected value) / (total number of values)]*100. Table 1 reflects the ranks (ID) and VALUES of *n* = 17 ordered numbers. To calculate the percentile for value 103, solve Percentile = [8/17)*100 = 47th percentile, so 47% of values are below 103. In this case, the data value is first specified, and the associated percentile is calculated.

ISSN: 2473-4901
v9 n5911

ID	VALUES
1	1
2	12
3	13
4	25
5	45
6	68
7	90
8	100
9	103
10	115
11	137
12	140
13	142
14	154
15	189
16	200
17	225

Table 1: Ranks and Values

On the other hand, the percentile value can be specified, and the associated data value can be located or calculated. This is the case in this tutorial. Percentile locations are based on ranks of ordered data. One may specifically want the 25th or 50th percentile value, corresponding to Q1 (First Quartile) or Q2 (Median), or the interquartile range along with the minimum and maximum values. Depending on whether there are an odd or even number of data values (*n*), the value at the *Pth* percentile will be located at a specific rank, or between two ranks. When between two ranks, the *Pth* percentile value is interpolated as an average of the values between the two ranks. This tutorial assumes an arithmetic average of values of ranked neighbors between the *R*th and *R*th+1 ranked locations. So, a percentile rank can be specified, and the corresponding value(s) located or calculated.

The equation for calculating the percentile rank differs depending on if *n* is odd or even. For *n* odd, Percentile Rank = [*p*]x[*n*+1], where *p* is the specified percentile value (in decimal notation) and *n* is the number of data values. For *n* even, Percentile Rank = [*p*]x[*n*]. For example, to find the 50th percentile value of the 17 odd number of values in Table 1, Percentile Rank = [.5]x[18] = 9th rank, corresponding to value 103. For the 25th Percentile Rank, Percentile Rank = [.25]x[18] = 4.5th rank. There is no 4.5th rank, so the values between ranks 4 and 5 are averaged; [25+45]/2 = 35. Note in Table 1, there is no value 35, so the 25th percentile value is an interpolated approximation. For a more thorough treatment of percentile calculations, see Frost, J. (2022).

4. THE MODULO OPERATOR

Determining if *n* is odd or even can be important in determining ranks and other SQL aggregates. SQL does not have a built-in function to determine if *n* is odd or even, so we introduce an arithmetic operator available in SQL to determine odd versus even *n* by finding the MODULO of two values; x MOD y. Other RDBMS use MODULO or % instead of MOD. The MOD function calculates the remainder of a value, the dividend (x), divided by another value, the devisor (y). So, if x/y = 8.1, the quotient is 8 and the remainder is 1. In this tutorial, our dividend is x = *n*, and our devisor will always be y = 2. Without a deep dive into MODULO calculus, accept that if *n* MOD 2 = 1, then *n* is odd, and if *n* MOD 2 = 0, then *n* is even. In our case, y is always chosen to be 2 to ensure that the MOD result is either 0 or 1. A quick glance of SQL **MOD** () with examples. EDUCBA, 2022.

Table 1 shows 17 data values, so *17* MOD *2* = 1, telling us that *n* is odd. This will be important when using an SQL projected IF function (**IIF**) to determine ranks based on if *n* is odd or even, as we can use the *n* **MOD** *2* operator in SQL. It should be noted that for large *n* with many data values falling into a few separate groups, the effect of *n* odd or *n* even on percentile calculations can be negligible, but still worth learning new skills in SQL.

5. PERCENTILES IN SQL

This tutorial introduces a method to locate or calculate percentile values in SQL for ungrouped data. That is, find the value in the data set corresponding to a specified percentile. Calculating percentile aggregates in SQL also allows additional calculations like the interquartile range, the median, and location of possible outliers.

The same technique can't be applied to grouped data. While ungrouped data are *n*-row data ordered in a database table, grouped data represent a rolled-up summary of the data, including group frequency counts, cumulative frequency, and cumulative percent frequency tabulations. Grouped data are commonly called a frequency distribution, or a frequency table. While it can be equally important to summarize grouped data, manuscript length restrictions prohibit introduction of applying this tutorial's

aggregates, with a goal demonstrating the method in a forthcoming tutorial.

While the proposed method uses many of the same common SQL keywords taught in an introductory database course, they diverge in the application of less commonly used textbook or tutorial-based SQL functions. The method requires use of basic **SELECT**, **FROM**, **WHERE**, and **AS** keywords, and further exemplifies using the **COUNT** aggregate, the **IIF** (comparison operator), the **MODULO** function, the **BETWEEN** operator, the **AND** operator, and the **AVG** (average) aggregate function. This method also exemplifies using SQL to provide a desired outcome of aggregates that are not available in any enterprise-level RDBMS, but may be useful and required for organizing and summarizing data using most common aggregates, or used in a data dashboard.

The method here is exemplified using Microsoft Office Access 2019/365, so there may be some variation in syntax with other RDBMS. For SQL statements in this tutorial, **BOLD ALL CAP** font is used for SQL **KEY WORDS**, **FUNCTIONS**, **IIF** statements, and **ARITHMATIC** and **COMPARISON** operators, as well as syntax characters including the comma, the parenthesis, and the semi-colon. *Italic* fonts are used for the names of *tables*, *queries*, and *attributes*, as well as user input *words*, *phrases*, and *numbers*. When querying from only one table or query, attribute names are surrounded in brackets, []. When querying from two or more tables or queries, [*table*].[*attribute*] and [*query*].[*attribute*] dot notation is used indicating the name of the table or query, a dot, and the attribute name. Attributes derived from aggregates, expressions and functions are aliased using the **AS** operator to improve the readability of the SQL.

6. METHOD FOR SQL IMPLEMNTATION TO CALCULATE PERCENTILES

The method is based on simple ranking of data depending on if the number of data values (*n*) is odd or even. Since the value *n* is used in SQL statements, it should be referenced as a dynamic value resulting from the SQL **SELECT COUNT**(*****) aggregate function, queried from the table of data. In this tutorial, the starting database table is always named *myData*, and includes a sorted ascending column of data values named *VALUES*, and a column named *ID* indexed from 1 to *n*, like 1, 2, 3, 4, 5 ….., *n*. The *ID* column must count from 1, associated with the smallest data value, to *n*, the largest data value. Before using this method, a one-time setup that can be recycled for any percentile calculation should to be completed, as shown below.

Complete the following 4 setup tasks.

Task 1: Create a table named *myData* and enter data from Table 1 into it, with columns named *ID* and *VALUES*. It should be identical to the data shown in Table 1, where *ID* is data type AutoNumber (integer), and *VALUES* is a data type number double. There is no primary key. See the table design in Appendix A.1.

Task 2: Create a table named *myPercentile* with one column named *P*, data type number double. This table will have one row, containing the decimal *Pth* percentile associated with the data value that is to be located or calculated. See the table design in Appendix A.2.

Task 3: Create a query named *myCount_n* using the SQL **SELECT COUNT** (*****) aggregate function as shown below to calculate the value *n* from the *myData* table. The count will be aliased **AS** *n*.

SQL
SELECT COUNT(*ID*) **FROM** *myData* **AS** *n***;**

Result
n = 17

Task 4: Create a query named *myMODULO* to calculate the result of *n* **MOD** 2 that will be used in two **IIF** operator projections. Use the SQL **SELECT** operator shown below to calculate the ***MOD***, resulting in either 0 or 1. The attribute name will be aliased **AS** *MOD*.

SQL
SELECT [*n*] **Mod** *2* **AS** *MOD* **FROM** *myCount_n***;**

Following the setup, this method has four simple steps shown below with an embedded example of finding the 50th percentile where *n* is odd.

Step 1: In the table *myPercentile*, enter the *Pth* percentile in decimal form. Note that *n* from the query *myCount_n*, and *MOD* from the query *myMODULO*, hold the data for input into the Step 2 query.

Example: Open table *myPercentile* and enter the value .5. Save and close the table.

Step 2: Write a query named *M1_myRanks* to determine the two ranks, *R1* and *R2*, associated with the *Pth* percentile. *R1'*s calculation necessarily depends on the values of *MOD* and *P*.

Locate the ranks *R1* and *R2* associated with the rank locations of the desired percentile. This query depends on a projection of two opening If statements (**IIF**) using the value of ***MOD*** from the query *myMODULO*. It then uses the value *P* from the *myPercentile* table to determine *R1* and *R2*.

Also note that *R2* is a function of *R1*, so no additional calculations are required for *R2*. The **IIF()** function returns a value if a condition is TRUE, or another value if a condition is FALSE. In this query, the values of *R1* and *R2* are based on the IIF function. In the first **IIF** statement, if *n* is odd (MOD = 1), then calculate the rank *R1* using [p]*[n+1], else calculate *R1* using [p]*[n]. In the second IIF statement, if *n* is odd, *R2* = *R1*, else R2 = R1+1. Notice that *R1* ranks are rounded to 0 decimal places, forcing integer ranks.

Example: SQL locating the two ranks.

SQL
```
SELECT
IIf([myMODULO].[MOD] = 1,
ROUND([myPercentile].[P]*([myCount_n].[n]+
1),0),
ROUND([myPercentile].[P]*[myCount_n].[n],0)
AS R1,
IIf([myMODULO].[MOD]=1, [R1],[R1]+1) AS
R2
FROM [myCount_n], [myPercentile],
[myMODULO];
```

Result Ranks
R1 = 9 and *R2* = 9

Step 3: Write a query named *M1_myValues* to locate the two values, *VALUE1* and *VALUE2*, that correspond to the two ranks determined in Step 2, *R1* and *R2*.

- Example: SQL locating the two values.

SQL
```
SELECT VALUES
FROM [myData], [M1_myRanks]
WHERE [myData].[ID]
BETWEEN [M1_myRanks].[R1] AND
[M1_myRanks].[R2];
```

Result Values
Value 1 = 103 and Value 2 = 103

Step 4: Write a query named *M1_myPercentileValue* using the SQL AVG () aggregate function to calculate the arithmetic mean of the two values from Step 3.

- Example: SQL uses the **AVG** aggregate function to average the two values (103 and 103), and the 50th percentile result,

SQL
```
SELECT AVG(VALUES) AS myPercentileValue
FROM M1_myValues;
```

Result Value
Value = 103, the 50th percentile

In the event there were 18 values, one can apply the same 4 steps to calculate the 50th percentile value. The next example will apply the same queries to an even number of data values; *n* = 18. Perform the following tasks.

Task 1: Open the *myData* table and append a row to the bottom; *ID* = 18, *VALUES* = 230.

Task 2: Run the query *M1_myRanks* and observe that the ranks are *R1* = 9 and *R2* = 10.

Task 3: Run the query *M1_myValues* and observe that the associated data values are 103 and 115.

Task 4: Run the query *M1_myPercentileValue* and observe that the median is 109, the average of 103 and 115.

For other percentile value calculations, one only need to input the desired percentile into the table *myPercentile*, then run the query *M1_myPercentileValue*.

In summary, the query progression follows.

Step 1: myPercentile - Enter Pth Percentile Value.
Step 2: M1_myRanks - Calculates Ranks to locate data values.
Step 3: M1_myValues - Locates Values to average.
Step 4: M1_myPercentileValue - Calculates Pth percentile value.

7. OBSERVATIONS

The method runs quickly, even for large data sets. In many cases, the determination of *n* being odd or even may have little effect on the output, but it has the added benefit of exposing SQL programmers to the lesser known MODULO function and using functions within the **IIF** domain to determine output values.

When the attribute data types are date values, the method returns serial numbers representing

the number of days that have elapsed since 01/01/1900. Although useful in date calculations, the output from the query might lose interpretive value unless converted back to a human friendly date representation. If dates were going to be typical inputs to the queries, one would need to edit the final query to cast the serial numbers as dates.

If percentile calculations were to be a common data summary, it would likely be best if an aggregate query were written that included, for example, the COUNT, AVG, MIN, Q1, Q2, Q3, MAX or any other percentile aggregate output. One may also consider aggregates for the variance and standard deviation, as well as the range and IQR. Using the IQR one could write a query to detect possible outliers. Any one of these desired outcomes may require a separate query, but the queries could be combined into a single query and set as a stored procedure.

Within the Microsoft Access environment, after the percentile value *P* is statically entered in the *myPercentile* table, all queries are dynamically executed like a stored procedure without user involvement. For other RDBMS, one may need to create a stored procedure that dynamically updates all queries based on a parameter query for the *P*th percentile. The benefit of a parameterized SQL query is that one can prepare it ahead of time and reuse it for similar applications without having to create distinct SQL queries for each case. Additionally, one may want to write a set of nested queries across all individual queries.

8. CONCLUSION

The purpose of the tutorial is to expand the skill set of those practicing or studying SQL to in higher level SQL experiences, especially in the increasing trend in SQL proficiency required by data-driven organizations. The higher-level skills presented are related to calculating percentiles for ungrouped data. Even without application to percentiles and other aggregates, SQL practitioners can still benefit from the value-added components of the tutorial, toward improved SQL skills, and higher-level understanding of SQL programming.

The higher skills went above the textbook examples and introduced lesser studied SQL like MODULO, and IFF, writing expressions for attributes, and formatting functions. The examples were presented in Microsoft Access 2019/365 since many users have Microsoft Access more readily available than an enterprise RDBMS. Appendices B to G relate to the teaching notes accompanying the tutorial. They include quizzes, large data set extraction and project completions. The completed Access database is available on request.

9. REFERENCES

Frost, J. (2022, March 13). *Percentiles: Interpretations and calculations*. Statistics By Jim. Retrieved November 20, 2022, from https://statisticsbyjim.com/basics/percentiles/#:~:text=To%20calculate%20an%20interpolated%20percentile,11%20%2B%201)%20%3D%208.4

SQL mod(): A quick glance of SQL mod() with examples. EDUCBA. (2022, June 10). Retrieved November 21, 2022, from https://www.educba.com/sql-mod/

SQL operators. (n.d.). Retrieved November 20, 2022, from https://www.w3schools.com/sql/sql_operators.asp

ISSN: 2473-4901
v9 n5911

APPENDIX A

Appendix A.1 – myData Table Data Types

myData

Field Name	Data Type	Description (Optional)
ID	Number	Integer Data Type
VALUES	Number	Double Data Type, Auto Decimal Places

Appendix A.2 - myPercentile Table Data Type

myPercentile

Field Name	Data Type	Description (Optional)
P	Number	Double Data Type, Auto Decimal Places