Extending SQL Implementation to Calculate Percentile/Rank Distributions and Percentile Aggregates

John N. Dyer
Georgia Southern University

ABSTRACT

As more and more organizations are data-driven, educators, students, and practitioners are often expected to possess adequate skills in basic Structured Query Language (SQL), and in many cases, skills beyond simple classroom/textbook applications. It is increasingly important that those in information systems, information technology, computer science (IS/IT/CS), and analytics possess a more robust skill set than what is learned in 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 might go back and forth between the database and external software to organize and summarize data; an export/analyze/import (EAI) approach. When real-time database summaries and aggregations are required, EAI is not an option. Unfortunately, any RDBMS provides only a small set of standard SQL aggregate functions such as the sum, count, average, minimum, maximum, variance, standard deviation, first, and last. Aggregates beyond these must be calculated using SQL in a programmatic approach. This paper presents and implements 3 query methods that allow learning and practicing non-SQL supported summaries/aggregates in real-time within the RDBMS, including percentile and rank distributions, calculations of exact and approximate percentiles, and a 5-number summary dashboard-type application. As such, these methods can be extended to other aggregates such as the interquartile range (IQR), measures of distribution skewness, and outlier detection. Hopefully, this paper can serve as a resource to facilitate higher learning skills and expand the capabilities of educators, students, and practitioners in SQL.

Keywords: Database, SQL, Aggregate Function, Percentile, CDF, Rank

INTRODUCTION

SQL has primarily been the domain of IS/IT/CS educators, students, and practitioners. Due to the rapid advancement in digitalization, there has been a shift wherein data-driven organizations depend heavily on database and data analytics practitioners possessing SQL as a top skill. As such, SQL has become ubiquitous in relational databases across most functional areas of modern organizations (Papiernik, M. (2022, October 19)). It has become imperative that a greater cross-section of educators, students, and practitioners possess more than just basic SQL skills (Dyer, John N. (2023)). Enki. (n.d.-a) makes the case that SQL is an essential skill in all areas of business, and that at best, SQL is glossed over in college courses, and since many developers do not work directly with SQL in their jobs, they do not have strong skills in SQL. And since SQL is the oldest, most established and widespread technology (in terms of data), SQL skills make developers more productive. Scores of "how-to" questions are posted online regarding more advanced queries within the RDBMS, like creating data distributions, percentile and rank calculations, and related aggregates. Those questions, and the methods in this paper have experienced little formal addressing in the literature, practice, or online forums.

This paper exemplifies SQL skills beyond the basic SELECT statement, allowing one to study and practice the breadth of query-based solutions available using SQL. The purpose is primarily to be a teaching case type application to enhance and expand the SQL skill sets for educators, students, and practitioners of SQL in academics and industry. Hopefully, the paper can help bridge the gap between classroom SQL and within-field practice, wherein many IS/IT/CS and analytics persons are expected to "hit-the-ground-running" in SQL. By experiencing SQL topics at a higher level through practice and case application, it is hoped that the student and practitioner can enhance their SQL skill set in terms of what more can be accomplished beyond basic SQL when one possesses skills providing solutions within the practice of SQL programming. It is hoped that this paper will make a meaningful contribution to such SQL skills. This paper is based on the work of Dyer, John N. (2023).

ACCESS DATABASE AS THE PLATFORM

Microsoft Office Access in the classroom has many benefits for the educator, student, and practitioner. Microsoft Access is taught in academics and frequently used in industry. Enlyft (n.d.-a) found that over 124,000 companies use Access, with most having between 50 and 200 employees, and annual revenue between $1 million $10 million. They identify industry users to include North Face, NetSuite and Red Hat, among many others. It is also estimated that Access has almost 11% market share (The Access Man. (n.d.)), (FMS (n.d.)). Additionally, DB-Engines (DB-engines ranking (n.d.)) places Access as the ninth most popular RDBMS worldwide. Access's ease of use, as well as prolific use in industry validates it as an appropriate RDBMS to use in teaching and practice.

Primarily, a course in database is to teach the basic concepts to design and implement a database, including database modeling, design, creating tables and relationships, and designing queries using built-in tools and SQL. A typical database course is usually not focused on any one RDBMS, such as Oracle, SQL Server, or MySQL, etc. All RDBMS offer some of the same basic functionality, but often with a different interface, development platform, scalability, and supporting software.

Unfortunately, teaching with enterprise level RDBMS often comes with high overhead costs including software installation and configuration, a steep learning-curve, excessive ”how-to” documentation and lab demonstrations, a high level of abstraction, sophisticated and sometimes difficult development tools and platforms, deployment, and remote accessibility issues, to name a few.

BASIC SQL AND AGGREGATE FUNCTIONS

Aggregate functions are calculations performed on a set of data resulting in a single number to accurately summarize the data. They are commonly used in descriptive statistics and analytics (Hayes, A. (n.d.)). SQL readily allows organizing and summarizing data within the domain of the SELECT and JOIN operators, projecting aggregate functions of data, calculated expressions, arithmetic and comparison operators, IF/CASE statements, and inline formatting (Dyer, John N. (2023)). A list of SQL operators is available at *SQL operators* (n.d). SQL includes a small set of built-in aggregate functions (sum, count, average, minimum, maximum, variance, standard deviation, first, last) but was not designed to calculate percentile or rank distribution summaries, percentile aggregates, descriptive aggregates like median, mode, range, IQR, skewness, business related indices, etc.

For these unsupported summaries and aggregates, one might employ the EAI approach. But, when the RDBMS is used to provide real-time data summaries/aggregation (SAP HANA/ABAP/Open SQL, Sales Force SQQL), such as used in managerial dashboards and visual summaries, EAI is not a workable approach, hence a reason to implement the non-SQL supported queries within the RDBMS. Many software applications in FinTech, marketing research, human resources, logistics, etc., are integrated with SQL platforms that require real-time applications of SQL, not simply EAI type API calls. Additionally, the EAI approach is a single-user instance, while in the domain of the RDBMS, the multi-user approach allows several users to share summary views and aggregates in real-time.

The queries implemented in this paper are important in providing a more 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 (Dyer, John N. (2023)). The queries include those to create a complete percentile/rank distribution of data (Section 7, Method 1), queries to calculate exact percentiles (Section 8, Method 2), and a query template to approximate percentiles that can be implemented to simultaneously calculate multiple percentiles including ventiles, deciles, quintiles, and quartiles (Section 9, Method 3).

PERCENTILES OVERVIEW

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 falls at or below it. Common percentiles include ventiles (5^{th} p, 10^{th} p, … 95^{th} p), deciles (10^{th} p, 20^{th} p, … 90^{th} p), quintiles (20^{th} p, 40^{th} p, 60^{th} p, 80^{th} p) and quartiles (25^{th} p, 50^{th} p, 75^{th} 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, calculate the value’s relative standing in the data set, even if the actual data value is unknown (Dyer, John N. (2023)). If one’s test score is at the 75^{th} percentile,

then 75% of the other scores fall at or below that score. Likewise, the interquartile range (IQR), calculated as the 75th percentile minus the 25th percentile (Q3-Q1), is a measure of variability (dispersion) for the middle 50 percent of data. The IQR is often used as a measure to calculate skewness and to detect outliers in data.

Percentile locations are based on ranks of ordered data. The percentile/rank distribution quantifies percentiles and ranks across all ordered data values. The percentile distribution is also known as the cumulative distribution function (CDF), and is widely used in empirical data analysis. The distribution shows the cumulative percentile of every value in the ordered data set. This is the case in the implementation in Method 1 below. In general, the percentile of a data value is given as

Percentile = [(count values at and below a selected value) / (count total values)]*100.

Dyer, John N. (2023) provided the Appendix A.1 table image reflecting the ID and data values for n = 17 ordered numbers, calculating the percentile for value 103. Solving for the Percentile = [9/17)*100 = 53rd, so 53% of values are at or below 103. In this case, the data value is specified first, and the associated percentile is calculated. On the other hand, the percentile value can be specified, and the associated data value can be located. One may want the 25th or 50th percentile value, corresponding to Q1 (1st Quartile) or Q2 (2nd Quartile/Median), or the IQR, along with the minimum and maximum values. This is the case in the implementation in Method 2 below.

When calculating aggregates, and depending on whether there are an odd or even number of data values (n), the value at the P^{th} percentile will be located at a specific rank, or between two ranks. When between two ranks, the P^{th} percentile value is interpolated as an average of the values between the two ranks, but other interpolation schemes exist (Dyer, John N. (2023)). The equation for calculating the exact percentile rank differs depending on if n is odd or even. When n is odd,

Percentile Rank (*odd*) = [p]x[n+1],

where p is the specified percentile value (in decimal notation) and n is the number of data values. When n is even,

Percentile Rank (*even*) = [p]x[n].

Dyer, John N. (2023) provided the example to find the 50th percentile value of the 17 values, Percentile Rank = [0.5]x[18] = 9th rank, corresponding to value 103. For the 25th percentile, Percentile Rank = [0.25]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. 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).

PERCENTILES IN SQL

This paper introduces 3 methods using SQL to calculate the entire percentile/rank distribution, to calculate exact percentiles, and to calculate approximate percentiles (based on the

percentile/rank distribution). While the methods implemented here use many of the common SQL keywords, they also apply less commonly used textbook-based or example-based SQL. The methods here require use of SELECT, FROM, WHERE, BETWEEN, AND, AS, and INSERT INTO keywords, and further exemplifies using the COUNT aggregate, the IIF statement, the MODULO function, the AVG (average) aggregate function, the TOP X PERCENT statement**,** and the FIRST and LAST aggregate statements.

SQL SYNTAX

Borrowing from Dyer, John N. (2023), for SQL syntax in this paper, **BOLD ALL CAP** font is used for SQL **KEY WORDS**, **FUNCTIONS**, **ARITHMATIC,** and **COMPARISON** operators, as well as syntax characters including the comma, parenthesis, and the semi-colon. *Italic* fonts are used for the names of *tables*, *queries*, and *columns*, as well as user input *words*, *phrases*, and *numbers*. When querying from only one table or query, column names are surrounded in brackets; []. When querying from two or more tables or queries, [*table*].[*column*] and [*query*].[*column*] dot notation is used, indicating the name of the table or query, a dot, and the column name. Columns derived from aggregates, expressions, and functions are aliased using the AS operator to improve the readability of the SQL.

METHOD 1 - SQL IMPLEMENTATION TO CALCULATE THE PERCENTILE/RANK DISTRIBUTION

Method 1 is based on an SQL **INSERT INTO** statement that inserts the raw data from the starting table (*myData*) into a table named *Percentile_Rank_Distribution*, calculates and inserts percentiles, ranks and the ID for all data values. If data values are equal they share the same rank, but each data value has a unique percentile. Complete the following 3 setup tasks to create the percentile/rank distribution.

Task 1: Create a table named *myData* and enter data shown in the Appendix A.1 table image. See the resulting table design shown in the Appendix A.2 table design image.

Task 2: Create a table named *Percentile_Rank_Distribution* with columns *ID* (Integer), *Percentile* (Double(Fixed), 4 decimal places), *Rank* (Integer), and *Data* (Double). See the resulting table design in the Appendix A.3 table design image.

Task 3: Create a query named *Percentile_Rank_Query* that inserts each data value's *ID*, *percentile* location, *rank* location, and *data* value into the *Percentile_Rank_Distribution* table. The result shown in the Appendix A.4 query image is the complete *ID*/*Percentile*/*Rank*/*Data* distribution.

The SQL to create the percentile/rank distribution follows.

SQL: *Percentile_Rank_Query*
INSERT INTO *Percentile_Rank_Distribution*
(*ID***,** *Percentile***,** *Rank***,** *Data***)**
SELECT [*myData*]**.**[*ID*]**,**

([*ID*]/(**SELECT COUNT**(*) **FROM** *myData*))***100 AS** *Percentile*,
(**SELECT COUNT**(*) **FROM** *myData* **AS** *D1* **WHERE** [*D1*].[*Data*] **<** [*myData*].[*Data*]) **+** *1*
AS [*Rank*], *Data* **FROM** *myData*;

The query can be recycled for different data sets by replacing the data in the *myData* table, deleting the existing data in the *Percentile_Rank_Distribution table*, and executing the query.

To interpret the SQL by line number, the program proceeds as follows.

Lines 1 and 2 use the INSERT INTO statement defining column names in parentheses.

Line 3 inserts the *ID* from the *myData* table into the query column *ID*.

Lines 4 and 5 calculate each percentile value by dividing each sequential *ID* number by the total count of data values (*n)*, then multiplies by 100, aliases it as *Percentile*, and inserts it into the query column *Percentile*.

Lines 6, 7, and 8 create an alias copy of the *myData* table named *D1*, then assigns a count for each row, where the data value in the *D1* alias is less than the data value in the *myData* table. The +1 forces ranks to start at 1. The projections are aliased as *Rank* and inserted into the query column *Rank*.

Line 8 inserts the data value from each row from the *myData* table, and uses the FROM statement to indicate the source table. In summary, the progression is as follows.

1. *myData*: Enter ordered data into the table.

2. *Percentile_Rank_Query*: Execute and view the *Percentile_Rank_Distribution* table.

METHOD 2 - SQL IMPLEMENTATION TO CALCULATE EXACT PERCENTILE AGGREGATES

Method 2 is based on simple ranking of data depending on whether the number of data values (*n*) is odd or even. Since the value *n* is used in SQL statements, it is referenced as a dynamic value using the SQL **SELECT COUNT**(*) aggregate function, queried from the table *myData*. Method 2 uses the arithmetic operator **MOD** (modulo) to determine if there is an odd or even number of data values. Other RDBMS use the keyword **MODULO** or the **%** symbol instead. SQL is used to determine odd versus even *n* by finding the modulo of two values; **x MOD y**. The **MOD** function calculates the remainder of a value, the dividend (x), divided by another value, the devisor (y). If for example x/y = 8.1, the quotient is 8 and the remainder is 1. In this implementation, the dividend is x = *n*, and the 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 this implementation, y is always chosen to be 2 to ensure that the MOD result is either 0 or 1.

For the *myData* table, *17* MOD *2* = 1, *n* is odd. This result is required in the implementation of Method 2. The **IFF** function is used to determine ranks based on if *n* is odd or

even. It is noted that all SQL platforms use **IF** functions, but Access does not allow the alternative **CASE** statement. As such, this implementation uses the **IFF** function as it is common across all SQL platforms. Before using this method, a one-time setup for any percentile calculation is completed, as shown below. First ensure the starting table *myData* from Method 1 Task 1 exists.

Complete the following 5 setup tasks.

Task 1: Create a table named *myPercentile* with one column named *P (Double).* The table has one row and one column, containing the decimal P^{th} percentile associated with the data value that is to be located. See the Appendix A.4 table design image.

Task 2: Create a query named *myCountModulo* using the SQL **SELECT COUNT** (*) aggregate function and *n* **MOD** *2* to calculate the count and modulo of data from the *myData* table. The count is aliased as *n* and the modulo is aliased as *Mod*.

SQL: *myCountModulo*
SELECT COUNT([ID]*)* **AS** *n*, *n* **MOD** *2* **AS** *Mod*
FROM *myData*;

Task 3: Create a query named *myRanks* to determine the two ranks, *R1* and *R2*, associated with the P^{th} percentile. This query depends on a two **IFF** functions using the value of *Mod* from the query *myCountModulo*. It then uses the value *P* from the *myPercentile* table to determine ranks *R1* and *R2*. The first **IIF** determines if *MOD* = 0 (*n* is even), then calculates the rank *R1* = [*P*]*[*n*+1], else *R1* = [*P*]*[*n*]. For the second **IIF**, if *n* is even, then *R2* = *R1*+1, else *R2* = *R1*. *R1* is rounded to 0 decimal places, forcing integer ranks.

SQL: *myRanks*
SELECT
IIf([*Mod*] = *0*,
Round([*P*]*([*n*] + *1*), *0*),
Round([*P*]*[*n*], *0*)) **AS** *R1*,
IIf([*Mod*] = *0*, [*R1*] + *1*], [*R1*]) **AS** *R2*
FROM *myCountModulo*, *myPercentile*;

Task 4: Create a query named *myValues* to locate the two values, *Value1* and *Value2*, that correspond to the two ranks in *myRanks*.

SQL: *myValues*
SELECT *Data*
FROM [*myData*], [*myRanks*]
WHERE [*myData*].[*ID*]
BETWEEN
[*myRanks*].[*R1*] **AND** [*myRanks*].[*R2*];

Task 5: Create a query named *myPercentileValue* using the SQL **AVG** aggregate function to calculate the arithmetic mean of the two values in *myValues*.

SQL: *myPercentileValue*
SELECT AVG([*Data*]) **AS** *myPercentileValue* **FROM** *myValues***;**

Method 2 Example Application

Following one-time setup, the queries are automatically updated each time a new set of data is entered into the *myData* table. The desired percentile is entered into the *myPercentile* table and the *myPercentileValue* query is executed. An example with results follows.

1. Open table *myPercentile* and enter the value *0.5*. Close the table.

2. Execute *myPercentileValue.*
   Result: *myPercentileValue* = 101.5

For the intermediate queries, one can execute each query below and view the results, although doing so is not necessary.

Query: *myCountModulo*
Results: *n* = 17, *Mod* = 1

Query: *myRanks*
Results: *R1* = 8, *R2* = 9

Query: *myValues*
Results: 100 and 103

The next example applies to an even number of data values; *n* = 18. Perform the following steps and view the results.

Step 1: Open the *myData* table and append a row to the bottom; *ID* = 18, *Data* = 230. Save and close the table.

Step 2: Execute the *myPercentileValue* query.
Result = *115*

In summary, the progression is as follows.

1. *myData*: Enter ordered data into the table.

2. *myPercentile*: Enter desired P^{th} percentile into the table.

3. *MyCountModulo*: Calculate the count *n* and modulo.

4. *MyRanks*: Locate the data ranks.

5. *MyValues*: Locate the data values.

6. *myPercentileValue*: Execute the query to calculate and view the P^{th} percentile value.

METHOD 3 - ALTERNATIVE SQL IMPLEMENTATION FOR PERCENTILE AGGREGATES

Method 3 allows one to create multiple queries from a query template using the *Percentile_Rank_Distribution* table as a source to approximate percentiles and ranks for pre-specified percentiles, such as the quartiles. One could also pre-specify the percentile/rank ventiles, deciles, and quintiles. The method is based on using the **TOP** *X* **PERCENT** SQL statement, querying the *Percentile_Rank_Distribution* table, and using the **LAST** SQL statement to locate the last row in the query.

The X in the **TOP** X **PERCENT** statement must be between 1 and 100, where 100 corresponds to the maximum data value. Common values of X include 25, 50, 75, corresponding to the quartile values. The advantage of this implementation is the ability to predefine as many individual approximate percentile calculations as desired, with relative ease, without being concerned with an even or odd number of data. As a whole, these percentiles can be used in an aggregate dashboard-type application to more fully summarize the distribution of data, as is shown in Appendix B.

The implementation requires *n* of at least 100. The resulting percentiles are very close approximations instead of exact percentiles in Method 2. The standard form of the query is shown below, named *P_Template*, which can be used as a template for any specified values of X. Note that the query is a nested query; a **SELECT** within a **SELECT**. The inner query returns the **TOP** X **PERCENT** of data values, while the outer query returns the **LAST** row of the inner query.

SQL: *P_Template*
SELECT
Last([*ID*]) **AS** *_ID***,**
Last([*Percentile*]) **AS** *_Percentile***,**
Last([*Rank*]) **AS** *_Rank***,**
Last([*Data*]) **AS** *_Data*
FROM
(SELECT TOP *X* **PERCENT** *ID***,** *Percentile***,** *Rank***,** *Data*
FROM *Percentile_Rank_Distribution*)**;**

The example below shows the implementation of the SQL to calculate the median, that is, Q2 (50^{th} percentile). Only X must be specified in the query line 7, **TOP** X **PERCENT**, shown below. The query name is *Q2*.

SQL: *Q2*
SELECT
Last([*ID*]) **AS** *_ID***,**

Last([*Percentile*]) **AS** *_Percentile*,
Last([*Rank*]) **AS** *_Rank*,
Last([*Data*]) **AS** *_Data*
FROM
(**SELECT TOP** *X* **PERCENT** *ID*, *Percentile*, *Rank*, *Data*
FROM *Percentile_Rank_Distribution*);

The aliased column names all start with the underscore, like *_ID*, since using the same column name as in the **LAST** statements create circular references. The results for the *Q2* query (*Q2* = 175.5) are shown in the Appendix A.6 query image, where *myData* has *n* = 131 records used in Method 1. The exact *Q2* value based on the Method 2 is 176.53. Appendix B provides the table structure (image B.1) and subsequent queries to create and populate a *Quartiles* table showing 5 rows of percentile, rank and data values for the minimum, Q1, Q2, Q3, and maximum. In statistics, this is called the 5-number summary. The example is used to show how the aggregate functions and summaries can be extended to a dash-board type summary aggregate of data.

CONCLUSION

The purpose of the paper is to enhance the skill set of those practicing or studying SQL toward a higher-level of learning and practice, especially in the increasing trend in SQL proficiency required by data-driven organizations. The skills presented are related to calculating percentile/rank distributions, as well as exact and approximate percentile aggregates for ungrouped data. The skills went beyond the limited textbook treatment of operators such as MOD, IF/CASE, TOP, FIRST, LAST statements, and introduced the percentile/rank distribution query, among others. The examples were presented in Microsoft Access since many users have Microsoft Access more readily available than an enterprise RDBMS. SQL educators, students and practitioners can benefit from the value-added components of the paper, toward improved SQL skills, and better understanding of SQL programming. The complete Access database with tables and queries is available upon request.

REFERENCES

Dyer, John N. (2023), Teaching Case. An Instructor's Tutorial and Student Project for Extending SQL Implementation to Calculate Percentile Aggregates for Ungrouped Data. Proceedings of the ISCAP Conference, Albuquerque, NM.,, pp 1-7, ISSN: 2473-4901v9n5911, https://iscap.us/proceedings/2023/index.html

DB-engines ranking (n.d.) historical trend of the popularity ranking of database management systems. Retrieved March 7, 2024 from https://db-engines.com/en/ranking_trend

Enki. (n.d.-a) Blog - why SQL is the #1 essential skill for all professionals. examples inside. Retrieved March 1, 2024 from https://www.enki.com/post/why-sql-is-the-1-essential-skill-for-all-professionals

Enlyft (n.d.-a) Microsoft Access Commands 7.77% market share in database management system. Microsoft Access Commands 7.7% market share in Database Management System in 2022. Retrieved March 4, 2024 from https://enlyft.com/tech/products/microsoft-access

FMS (n.d.) Microsoft Access within an organization's overall database strategy. Microsoft Access within an Organization's Database Strategy. Retrieved March 1, 2024 from http://www.fmsinc.com/MicrosoftAccess/Strategy/index.asp

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

Hayes, A. (n.d.). Aggregate function: Definition, examples, and uses. Investopedia. Retrieved March 7, 2024 from https://www.investopedia.com/terms/a/aggregate-function.asp

Isarocket (2024a, January 30). Microsoft Migration: Why you should upgrade from access. InterSoft Associates Custom Software Solutions. Retrieved March 7, 2024 from https://intersoftassociates.com/articles/migrations/microsoft-migration-why-you-should-upgrade-from-access/#:~:text=Did%20You%20Know:,is%20available%20in%2026%20languages.

Papiernik, M. (2022, October 19). Why you should learn SQL. DigitalOcean. Retrieved March 1, 2024 from https://www.digitalocean.com/community/conceptual-articles/why-you-should-learn-sql

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

APPENDIX A

Appendix A.1 – myData

myData

ID	Data
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

Appendix A.2 – myData Table Columns and Data Types

myData

Field Name	Data Type	Description (Optional)
ID	Number	Integer Data Type
Data	Number	Double Data Type

Appendix A.3 – Percentile Rank Distribution Table Columns and Data Types

Percentile_Rank_Distribution

Field Name	Data Type	Description (Optional)
ID	Number	Integer Number Data Type
Percentile	Number	Fixed Number Data Type, 4 Decimal Places
Rank	Number	Integer Data Type
Data	Number	Double Data Type

Appendix A.4 – Percentile Rank Distribution

Percentile_Rank_Distribution

ID	Percentile	Rank	Data
1	5.8824	1	1
2	11.7647	2	12
3	17.6471	3	13
4	23.5294	4	25
5	29.4118	5	45
6	35.2941	6	68
7	41.1765	7	90
8	47.0588	8	100
9	52.9412	9	103
10	58.8235	10	115
11	64.7059	11	137
12	70.5882	12	140
13	76.4706	13	142
14	82.3529	14	154
15	88.2353	15	189
16	94.1176	16	200
17	100.0000	17	225

Appendix A.5 - myPercentile Table Data Type

myPercentile

Field Name	Data Type	Description (Optional)
P	Number	Double Data Type

Appendix A.6 – Q2 Query Results

Q2

_ID	_Percentile	_Rank	_Data
66	50.381679389313	66	175.5

APPENDIX B

Setup Tasks

1. Insert data into the *MyData* table (*n* >= 100).
2. Delete existing data in the *Percentile_Rank_Distribution* table.
3. Execute the *Percentile_Rank* query.

Complete the 3 steps below.

Step 1: Create a table named *Quartiles* with columns *ID*, *Percentile*, *Rank*, *Date*, as shown in the Appendix B.1 table design image below. The percentile column can use data type Double (fixed) with desired number of decimal places.

Appendix B.1 - Quartiles

Quartiles

Field Name	Data Type	Description (Optional)
ID	Number	Integer Data Type
Percentile	Number	Double Data Type
Rank	Number	Integer Data Type
Data	Number	Double Data Type

Step 2: Create the 5 queries below. The first query is for the minimum, followed by Q1, Q2, and Q3, ending with a query for the maximum. Note that the *Q0_Insert* query can use **MIN** instead of **FIRST**, and the *Q4_Insert* query can use **MAX** instead of **LAST**.

SQL: Q0_Insert (Minimum)
INSERT INTO *Quartiles* (*ID*, *Percentile*, *Rank*, *Data*)
SELECT FIRST([*ID*]) **AS** *_ID*, **FIRST**([*Percentile*]) **AS** *_Percentile*, **FIRST**([*Rank*]) **AS** *_Rank*, **FIRST**([*Data*]) **AS** *_Data*
FROM *Percentile_Rank_Distribution*;

SQL: Q1_Insert (Q1/25th Percentile)
INSERT INTO *Quartiles* (*ID*, *Percentile*, *Rank*, *Data*)
SELECT [*_ID*], [*_Percentile*], [*_Rank*], [*_Data*]
FROM *Q1*;

SQL: Q2_Insert (Q2/50th Percentile)
INSERT INTO *Quartiles* (*ID*, *Percentile*, *Rank*, *Data*)
SELECT [*_ID*], [*_Percentile*], [*_Rank*], [*_Data*]
FROM *Q2*;

SQL: Q3_Insert (Q3/75th Percentile)
INSERT INTO *Quartiles* (*ID*, *Percentile*, *Rank*, *Data*)
SELECT [*_ID*], [*_Percentile*], [*_Rank*], [*_Data*]
FROM *Q3*;

SQL: Q4_Insert (Maximum)
INSERT INTO Quartiles (*ID*, *Percentile*, *Rank*, *Data*)
SELECT LAST([*ID*]) **AS** *_ID*, **LAST**([*Percentile*]) **AS** *_Percentile*, **LAST**([*Rank*]) **AS** *_Rank*, **LAST**([*Data*]) **AS** *_Data*
FROM *Percentile_Rank_Distribution*;

Step 3: Execute each of the 5 queries in sequence.

The Appendix B.2 table image displays the results in the table *Quartiles* based on Section 8, Method 1 example using $n = 131$ data values.

Appendix B.2 – Quartiles Results

Quartiles

ID	Percentile	Rank	Data
1	5.88	1	1
33	25.19	33	50
66	50.38	66	175.5
99	75.57	99	235.2
131	100.00	131	288