SQL Functions and Joins

SQL Functions

- An **aggregate function** performs a calculation on a set of values and returns a single value.
- The basic aggregate functions are the following:
  - **MIN** – the minimum attribute value encountered in a given column
  - **MAX** – the maximum attribute value encountered in a given column
  - **COUNT** – the number of rows containing non-null values
  - **SUM** – the sum of all values for a given column
  - **AVG** – the arithmetic mean (average) for a specified column
- *Tables for Examples*

Table 1. Customers

CustomerID	LastName	FirstName	ContactNum
3425	Reyes	Mark	09171056453
3446	Velasco	Marco	09167614778
3458	Santos	Markus	09565039378
3464	dela Cruz	Miguel	09195341063
3472	Cruz	Martin	09171195710

Table 2. Orders

OrderID	OrderDate	Amount	CustomerID
1	2020-02-14	130.50	3446
2	2020-02-14	297.50	3425
3	2020-02-14	794.25	3472
4	2020-02-15	332.75	3420

- **MIN**
  - *Syntax*: **SELECT MIN***(column_name)* **FROM** *table_name***;**
  - *Example:* SELECT MIN(Amount) FROM Orders;
  - *Result:* 130.50
- **MAX**
  - *Syntax*: **SELECT MAX***(column_name)* **FROM** *table_name***;**
  - *Example:* SELECT MAX(Amount) FROM Orders;
  - *Result:* 794.25
- **COUNT**
  - *Syntax*: **SELECT COUNT***(column_name)* **FROM** *table_name***;**
  - ***Example 1****:* SELECT COUNT(OrderID) FROM Orders;
    Result: 4
  - ***Example 2****:* SELECT COUNT(OrderID) FROM Orders WHERE Amount > 300;
    Result: 2
- **SUM**
  - *Syntax*: **SELECT SUM***(column_name)* **FROM** *table_name***;**
  - *Example:* SELECT SUM(Amount) FROM Orders;
  - *Result:* 1553.75
- **AVG**
  - *Syntax*: **SELECT AVG***(column_name)* **FROM** *table_name***;**
  - *Example:* SELECT AVG(Amount) FROM Orders;
  - *Result:* 388.4375
- The **GROUP BY** command option is often used with aggregate functions to group the result-set by one or more columns.
  - *Syntax*: **SELECT** *columns* **FROM** *table_name* **GROUP BY** *columns*;
  - *Example:* SELECT OrderDate, COUNT(OrderID) FROM Orders GROUP BY OrderDate;
  - *Result:*

OrderDate	(No column name)
2020-02-14	3
2020-02-15	1

Use an **alias** to assign a temporary name to the column. It can also be given to tables.

  - *Syntax*: **SELECT** *column_name* AS *alias* **FROM** *table_name*;
  - *Example:* SELECT OrderDate, COUNT(OrderID) AS Quantity FROM Orders GROUP BY OrderDate;
  - *Result:*

OrderDate	Quantity
2020-02-14	3
2020-02-15	1

- The **HAVING** command option is used to restrict the output of a GROUP BY query by applying conditional criteria to the grouped rows.
  - *Syntax*: **SELECT** *columns* **FROM** *table_name* **GROUP BY** *columns* **HAVING** *condition*;
  - *Example:* SELECT OrderDate, COUNT(OrderID) AS Quantity FROM Orders GROUP BY OrderDate HAVING Quantity > 2;
  - *Result:*

OrderDate	Quantity
2020-02-14	3

- The most commonly used date functions are as follows:
  - **YEAR** – returns a four-digit year
  - **MONTH –** returns the number of the month
  - **DAY** – returns the number of the day
    Example: SELECT OrderID, YEAR(OrderDate) AS Year, MONTH(OrderDate) AS Month, DAY(OrderDate) AS Day FROM Orders;
    Result of query:

OrderID	Year	Month	Day
1	2020	2	14
2	2020	2	14
3	2020	2	14
4	2020	2	15

  - **GETDATE** – returns the current date and time
    Syntax: **SELECT GETDATE();**
  - **DATEADD –** adds a number of selected time/date periods to a date then returns the date
    Syntax: **SELECT DATEADD(datepart, number, date);**
    Example 1*:* SELECT DATEADD(month, 2, '2020/12/14');
    Result: 2021-02-14 00:00:00.000
    Example 2*:* SELECT DATEADD(quarter, 2, '2020/02/14');
    Result: 2020-08-14 00:00:00.000
    Example 3*:* SELECT DATEADD(month, -2, '2020/02/14');
    Result: 2019-12-14 00:00:00.000
    Example 4*:* SELECT DATEADD(hour, 2, DATEADD(minute, 30, '2020/02/14'));
    Result: 2020-02-14 02:30:00.000
  - **DATEDIFF** – returns the difference between two (2) dates
    Syntax: **DATEDIFF(datepart, date1, date2);**
    Example: SELECT DATEDIFF(month, '1989-07-25', '2020-02-14');
    Result: 367
- The following are some of the most used numeric functions:
  - **ABS** – returns the absolute value of a number
    Example: SELECT ABS(-234.5);
    Result: 234.5
  - **ROUND** – rounds a number to a specified number of decimal places
    Syntax: **ROUND***(number, decimal places)*;
    Example 1*:* SELECT ROUND(234.459, 1);
    Result: 234.500
    Example 2*:* SELECT ROUND(234.459, 2);
    Result: 234.460
  - **CEILING** – returns the smallest integer value that is greater than or equal to a number
    Example: SELECT CEILING(234.1);
    Result: 235
  - **FLOOR -** returns the largest integer value that is less than or equal to a number
    Example: SELECT FLOOR(234.5);
    Result: 234
- The most commonly used string functions are as follows:
  - **CONCAT** – joins two or more strings together
    Syntax: **CONCAT(***string1, string2, …);*
    Example: SELECT CONCAT(FirstName + ' ' , LastName) FROM Customers WHERE CustomerID = 3446;
    Result: Marco Velasco
  - **LOWER** – returns a string in lowercase letters
  - **UPPER –** returns a string in all capital letters
    Example: SELECT UPPER('sql') + ' ' + LOWER('FUNCTIONS');
    Result: SQL functions
  - **SUBSTRING** – returns a part of a string
    Syntax: **SUBSTRING***(string, start position, length)*;
    Example: SELECT SUBSTRING('SQL Functions', 1, 3);
    Result: SQL
  - **LEN** – returns the number of characters in a string
    Example 1: SELECT LEN('SQL Functions');
    Result: 13
    Example 2: SELECT LEN('SQL Functions ');
    Result: 13
  - **TRIM** – removes the spaces or specific characters from start or end of a string
    Syntax: **TRIM**([*characters* FROM]*string*);
    Example: SELECT TRIM('<!> ' FROM ' <Functions! >');
    Result: Functions

SQL Joins

- A **JOIN** clause combines rows from two or more tables based on a common column.
- The types of join are as follows:
  - **INNER JOIN** – returns rows that have matching values in both tables

  Outer Joins
  - **LEFT JOIN** – returns all rows from the left table and the matched records from the right table. If there is no match from the right side, the result is NULL.
  - **RIGHT JOIN** – returns all rows from the right table and the matched records from the left table. If there is no match from the left side, the result is NULL.
  - **FULL JOIN** – returns all records when there is a match on either the left or right table. If there is no match, the missing side will contain NULL.
- **INNER JOIN**
  - *Syntax*: **SELECT** *columns* **FROM** *table1* **INNER JOIN** *table2* **ON** *table1.column_name = table2.column_name;*
  - *Example*: SELECT Customers.CustomerID, Customers.FirstName, Orders.OrderID FROM Customers INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
  - *Result of query:*

CustomerID	FirstName	OrderID
3425	Mark	2
3446	Marco	1
3472	Martin	3

- **LEFT JOIN**
  - *Syntax*: **SELECT** *columns* **FROM** *table1* **LEFT JOIN** *table2* **ON** *table1.column_name = table2.column_name;*
  - *Example*: SELECT Customers.CustomerID, Customers.FirstName, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
  - *Result of query:*

CustomerID	FirstName	OrderID
3425	Mark	2
3446	Marco	1
3458	Markus	NULL
3464	Miguel	NULL
3472	Martin	3

- **RIGHT JOIN**
  - *Syntax*: **SELECT** *columns* **FROM** *table1* **RIGHT JOIN** *table2* **ON** *table1.column_name = table2.column_name;*
  - *Example*: SELECT Customers.CustomerID, Customers.FirstName, Orders.OrderID FROM Customers RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
  - *Result of query:*

CustomerID	FirstName	OrderID
3446	Marco	1
3425	Mark	2
3472	Martin	3
NULL	NULL	4

- **FULL JOIN**
  - *Syntax*: **SELECT** *columns* **FROM** *table1* **FULL OUTER JOIN** *table2* **ON** *table1.column_name = table2.column_name;*
  - *Example*: SELECT Customers.CustomerID, Customers.FirstName, Orders.OrderID FROM Customers FULL JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
  - *Result of query:*

CustomerID	FirstName	OrderID
3425	Mark	2
3446	Marco	1
3458	Markus	NULL
3464	Miguel	NULL
3472	Martin	3
NULL	NULL	4

References:

Coronel, C. and Morris, S. (2017). *Database systems: design, implementation, and management* (12th ed.). USA: Cengage Learning.

Elmasri, R. and Navathe, S. (2016). Fundamentals of database systems (7th ed.). USA: Pearson Higher Education.

Kroenke, D. and Auer, D. (2016). Database processing: fundamentals, design, and implementation. England: Pearson Education Limited.