EXPLORE AI ACADEMY

SQL numeric functions

An introduction to SQL functions

What are SQL functions?

> SQL functions are **built-in operations** that can be used to perform various **calculations**, **manipulations**, or **transformations** on data within a **SQL database**. Advantages of using these functions include:

01. Efficiency

They take advantage of **internal algorithms** and **data structures** which results in faster query execution times and improved overall performance.

02. Reliability and compatibility

They adhere to the **SQL standards** and are **implemented consistently** across different database platforms.

03. Documentation and built-in support

They come with **extensive documentation** provided by the database management system vendors, making them **easier for developers to understand and use** effectively.

04. Extensive functionality

Database management systems provide a **wide range of built-in functions** that allow for complex calculations, string manipulations, and date and time operations.

05. Portability

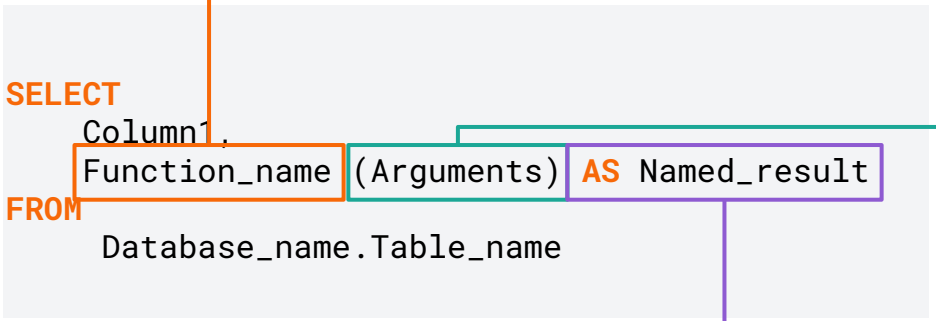
SQL queries use **standard functions**, therefore they can be **easily migrated or executed** on different database systems without significant modifications.

06. Ease of use

They are **readily available for use** without any additional configuration. This saves you from reinventing the wheel by implementing common operations from scratch.

Syntax of SQL functions

The **output of a function** in SQL is treated as a **derived column** and appears in the `SELECT` clause of a query. The general syntax of an SQL function is:



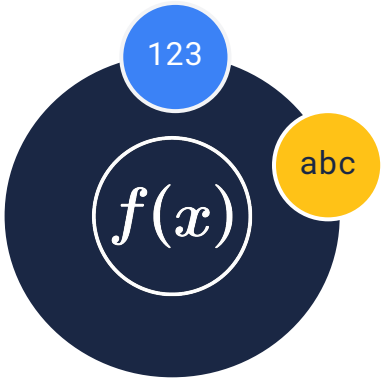
This defines the **specific SQL function** we want to use, such as `SUM` or `COUNT`.

These are the **input arguments** that the function requires. Some functions do not require any arguments while others require one or more arguments.

It is good practice to **name function results** descriptively with prefixes that explain the function, e.g. total_sales for `SUM(sales)`.

Function types

Functions can be categorised by **what input** they take.



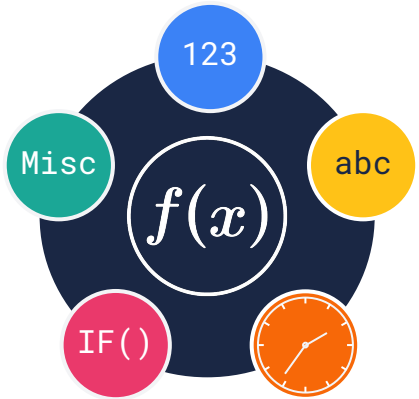
Numeric

These functions perform **calculations on a set of values** in a column and **return a numerical value**.
`SUM()`, `AVG()`, `COUNT()`, `MAX()`, `MIN()`, `POWER()`, `SQRT()`, `ROUND()`

String

These functions **operate on string values** and perform operations such as concatenation, manipulation, and text formatting.
`CONCAT()`, `LENGTH()`, `SUBSTRING()`, `UPPER()`, `LOWER()`

Function types



Many functions do not fit this classification! For example, `MIN()` works with strings, numbers, and dates. Focus on **understanding** how **each function** works, rather than classifying them.

Datetime

These are functions used to **handle date and time values** and perform operations like formatting, extraction, and manipulation.
`CURRENT_DATE()`, `DATEADD()`, `FORMAT()`

Conditional flow

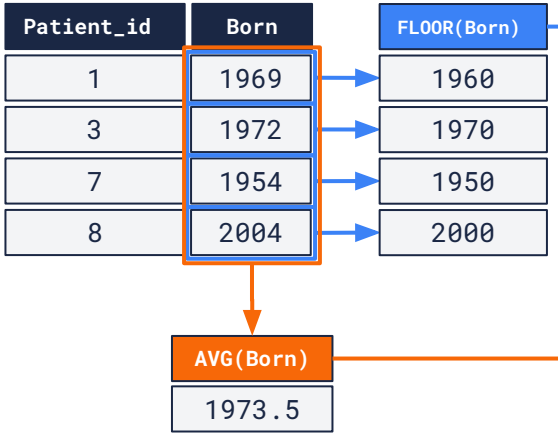
These functions allow for conditional logic in your SQL queries.
`CASE()`, `IF()`, `IIF()`

Miscellaneous

This category represents a variety of functions that **do a variety of things** such as converting data types and dealing with `NULL` values.
`CONVERT()`, `CAST()`, `NULLIF()`, `IFNULL()`

How functions behave

SQL functions vary in behaviour. **Aggregate functions** summarise data at a **column level**. **Scalar functions** manipulate data at the **row level**.



Scalar functions take a row and return a single **value for each row**, e.g. `UPPER()`, `LOWER()`, `CONVERT()`, `CAST()`, `IF()`, `FORMAT()` and arithmetic operators () `+`, `-`, `*`, `/`

Example:
`FLOOR(Born)` rounds the values in each row down to the nearest integer. **Each row** has a **new** decade **column** with the output of `FLOOR()`.

Aggregate functions take a **set of rows** as input and return a **single summary value**, e.g. `SUM()`, `AVG()`, `COUNT()`, `MAX()`, `MIN()`

Example:
`AVG(Born)` calculates the average of the values in `born`, and returns a single summary value of the column.

Window functions are a third way in which functions can behave which is a "hybrid" of **aggregate** and **scalar** functions.

Example data

To find the **total amount of income expenditure** in the **Free State province** in South Africa, we are going to use the South African Household Income and Expenditure Survey dataset (SAHIES).

The table is named **`Income_expenditure_2020`** in the `Sahies` database:

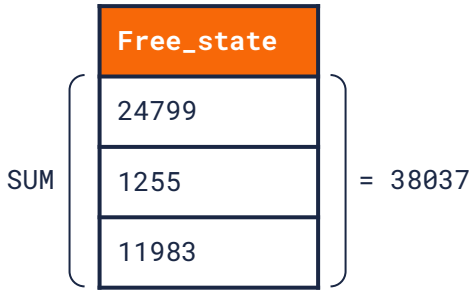
Expenditure_group	Western_cape	Northern_cape	Free_state	Kwazulu_natal	North_west	…	Mpumalanga	Limpopo
Housing	16400	20000	24799	21200	21200	…	22799	23599
Recreation	1521	989	1255	1217	1179	…	1065	912
Transport	17974	15406	11983	13481	12625	…	12839	13481

Example: Aggregate functions

> `SUM()` is an **aggregate function** that returns the total sum of a numeric column.

Query

```
SELECT
    SUM(Free_state) AS Total_free_state
FROM
    Sahies.Income_expenditure_2020;
```



Output

Total_free_state
38037

We see that our query returns a **single value** which is the sum of all rows of the **`Free_state`** column.

Example: Multiple aggregate functions

We can aggregate **multiple columns** or the same **column** using a **different function**.

Suppose we want to **calculate the total income spent** for the **Western** and **Free State** provinces and the **average spent** for the **Free State** province.

Query

```
SELECT
    SUM(Western_cape) AS Total_spent_western_cape,
    SUM(Free_state) AS Total_spent_Free_state,
    AVG(Free_state)
FROM
    Sahies.Income_expenditure_2020;
```

By not naming the average spent in the Free State, it is assigned a default name that may not be as descriptive.

Output

Total_spent_western_cape	Total_spent_Free_state	AVG(Free_state)
35895	38037	12679.0000

Example: Multiple aggregate functions

> Since aggregate functions produce a single row, we **cannot use** them with the columns in our main table.

Query

```
SELECT
    Free_state,
    AVG(Free_state)
FROM
    Sahies.Income_expenditure_2020;
```

Free_state has 3 rows, and `AVG(Free_state)` has 1, so running this query results in an error.

Output

Error!

Example: Scalar functions

> Suppose we want to calculate the **difference in income expenditure between** the **Free State** and the **Northern Cape** province.

Query

```
SELECT
      Expenditure_group,
      (Free_state
      - Northern_cape) AS Diff_fs_and_nc
FROM
    Sahies.Income_expenditure_2020;
```

Free_state		Northern_cape
24799	−	20000
1255	−	989
11983	−	15406

Output

Expenditure_group	Diff_fs_and_nc
Housing	4799
Recreation	266
Transport	-3423

Note how this name is slightly harder to understand because we used abbreviations.

We see that the **calculation occurred on a row level** because the results set includes a value for each row.

Using functions together

We can use **functions within functions**, known as **nesting**. SQL evaluates the **innermost** function **first** and then works its way **outwards**. The result of each inner function is used as the input for the outer function.

Query

```
SELECT
    SUM(Western_cape) AS Total_spent_western_cape,
    SUM(Northern_cape) AS Total_spent_northern_cape,
    ROUND(AVG(Free_state),0) AS Average_spent_free_state
FROM
    Sahies.Income_expenditure_2020;
```

ROUND() rounds a value to the specified amount of decimals. `AVG(Free_state)` is calculated first, then rounded to `0` decimal places.

Output

Total_spent_western_cape	Total_spent_northern_cape	Average_spent_free_state
35895	36395	12679

SQL functions without input arguments

> Some SQL built-in functions **do not require any arguments**. They are used to perform calculations or retrieve information.

CURRENT_DATE()

Returns the current date.

Query:

```
SELECT
    CURRENT_DATE();
```

Output:

CURRENT_DATE()
2023-06-20

RAND()

Generates a random number between 0 and 1.

Query:

```
SELECT
    Free_state,
    RAND() AS Random_number FROM…;
```

Output:

Limpopo	Random_number
23599	0.81247…
912	0.01824…
13481	0.55863…