Built-In Functions

Contents

[1. Number-Related Functions 3](#_Toc186546240)

[a) Round Function: 3](#_Toc186546241)

[b) Truncate Function: 5](#_Toc186546242)

[c) Ceiling Function: 6](#_Toc186546243)

[d) Floor Function**:** 6](#_Toc186546244)

[Differences Between Functions: 6](#_Toc186546245)

[2. String-Related Functions 7](#_Toc186546246)

[a) Length Function 7](#_Toc186546247)

[b) Upper and Lower Functions 7](#_Toc186546248)

[c) Trim Functions 7](#_Toc186546249)

[d) Locate Function 7](#_Toc186546250)

[e) Substring Functions 8](#_Toc186546251)

[f) Best Practices 8](#_Toc186546252)

[3. Date-Related Functions 9](#_Toc186546253)

[Getting Current Date and Time 9](#_Toc186546254)

[Fetching Current Date or Time Separately 9](#_Toc186546255)

[Extracting Components from a Date 9](#_Toc186546256)

[Adding or Subtracting Intervals 10](#_Toc186546257)

[Calculating the Difference Between Dates 10](#_Toc186546258)

[4. IFNULL**,** COALESCE**,** andIF-ELSE 11](#_Toc186546259)

[Handling Null Values with IFNULL: 11](#_Toc186546260)

[Handling Multiple Conditions with COALESCE: 12](#_Toc186546261)

[IF-ELSE: 13](#_Toc186546262)

[CASE statement 13](#_Toc186546263)

# Number-Related Functions

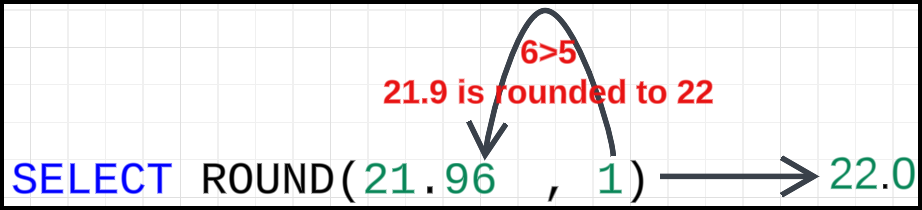
## **Round Function**:

* **Usage**:

SELECT ROUND(NUMBER, PRECISION AFTER DECIMAL)

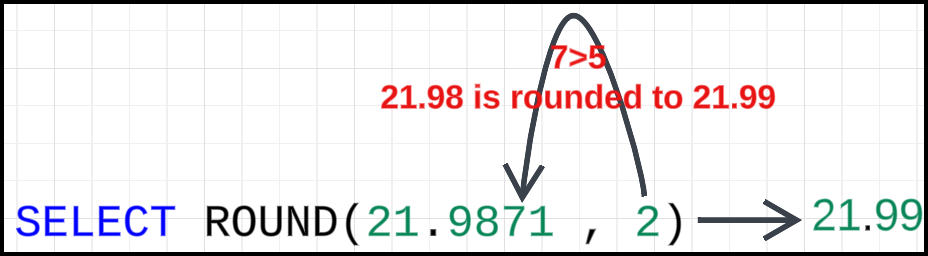
* Rounds a number to the specified number of decimal places (precision).
* If the digit after the specified precision is 5 or more, it rounds up; otherwise, it rounds down.
* Examples:





### Explanation:

1. **Understanding the Input:**
   * The number 21.96 has **2 digits** after the decimal point.
   * The precision specified (1) means we want the result to have only **1 digit** after the decimal point.
2. **Steps in the Rounding Process:**
   * **Step 1:** Identify the digit at the precision point.
     + Precision point = **1st digit after the decimal**.
     + In 21.96, the **1st digit after the decimal** is **9**.
   * **Step 2:** Look at the digit immediately **after** the precision point.
     + The digit after the precision point = **6** (2nd digit after the decimal).
   * **Step 3:** Apply rounding rules:
     + If the digit after the precision point (6) is:
       - 0–4: Do not increment the precision digit (round down).
       - 5–9: Increment the precision digit by 1 (round up).
     + Here, the digit is **6**, so we increment the precision digit (**9**) by 1.
   * **Step 4:** Handle the carry-over:
     + Incrementing **9** by 1 results in **10**.
     + The digit 10 rolls over, increasing the integer part of the number by 1.
     + The result becomes **22.0**.
3. **Result:**
   * The number 21.96, rounded to 1 decimal place, is 22.0.



### Explanation:

1. **Understanding the Input:**
   * The number 21.9871 has 4 digits after the decimal point.
   * The precision specified (2) indicates that the result should retain only **2 decimal places**.
2. **Steps in the Rounding Process:**
   * **Step 1:** Identify the digit at the precision point.
     + Precision point = **2nd digit after the decimal**.
     + In 21.9871, the second digit after the decimal is **8**.
   * **Step 2:** Look at the digit immediately **after** the precision point.
     + Digit after the precision point = **7** (3rd digit after the decimal).
   * **Step 3:** Apply rounding rules:
     + If the digit after the precision point (7) is:
       - 0–4: Do not increment the precision digit (round down).
       - 5–9: Increment the precision digit by 1 (round up).
     + Here, the digit is **7**, so we increment the precision digit (8) by 1.
   * **Step 4:** Update the number:
     + Incrementing 8 by 1 gives **9**.
     + The result becomes 21.99.
3. **Result:**
   * The number 21.9871, rounded to 2 decimal places, is 21.99.

## **Truncate Function**:

* **Usage**:

SELECT TRUNCATE(NUMBER, PRECISION AFTER DECIMAL)

* Truncates a number to the specified number of decimal places by simply removing the extra digits.
* No rounding occurs; digits after the specified precision are discarded.
* Examples:

SELECT TRUNCATE(21.96, 1) → 21.9

SELECT TRUNCATE(21.9871, 2) → 21.98

## **Ceiling Function**:

* **Usage**:

SELECT CEILING(NUMBER);

* Returns the smallest integer greater than or equal to the given number.
* Always rounds up, even for fractional values close to zero.
* Examples:

SELECT CEILING(21.1) → 22

SELECT CEILING(21.0001) → 22

## **Floor Function:**

* **Usage**:

SELECT FLOOR(NUMBER)

* Returns the largest integer less than or equal to the given number.
* Always rounds down.
* Examples:

SELECT FLOOR(21.9) → 21

SELECT FLOOR(21.0001) → 21

## Differences Between Functions:

* **Round** approximates to the nearest value based on the precision.
* **Truncate** removes digits without rounding.
* **Ceiling** always rounds up to the nearest integer.
* **Floor** always rounds down to the nearest integer.

# String-Related Functions

## **Length Function**

* **Purpose**: Finds the length of a string.
  + **Example**:

SELECT LENGTH('Delhi') → Output: 5.

## **Upper and Lower Functions**

* **Upper(x)**: Converts the string x to uppercase.
  + Example:

SELECT UPPER('delhi') → Output: DELHI.

* **Lower(x)**: Converts the string x to lowercase.
  + Example:

SELECT LOWER('DELHI') → Output: delhi.

* **Use Case**: Ensures uniformity in outputs.

## **Trim Functions**

* **Purpose**: Removes extra spaces from a string.
* **Types**:
  + **LTRIM(x)**: Removes spaces from the **start** of the string.

SELECT LTRIM(' I am going to Delhi ') → Output: 'I am going to Delhi '.

* **RTRIM(x)**: Removes spaces from the **end** of the string.

SELECT RTRIM(' I am going to Delhi ') → Output: ' I am going to Delhi'.

* **TRIM(x)**: Removes spaces from **both start and end**.

SELECT TRIM(' I am going to Delhi ') → Output: 'I am going to Delhi'.

## **Locate Function**

* **Purpose**: Finds the position of a substring within a string.
* **Syntax**: LOCATE(substring, string)
* **Output**: Returns the **index of the first occurrence** of the substring (indexing starts from 1).
  + Example:
    - SELECT LOCATE('P', 'Chopsy'); → Output: 4.
    - Returns 0 if the substring does not exist.
* **Use Case**: Similar to indexOf in programming languages.

## **Substring Functions**

* **Purpose**: Extracts a part of the string.
* **Types**:
  + LEFT(string, n):
    - Retrieves the first n characters from the **start** of the string.
      * Example: SELECT LEFT('Delhi', 2) → Output: De.
  + RIGHT(string, n):
    - Retrieves the last n characters from the **end** of the string.
      * Example: SELECT RIGHT('Delhi', 2) → Output: hi.
  + SUBSTR(string, start\_index, length):
    - Extracts a substring starting from start\_index (1-based) for a specified length.
      * Example: SELECT SUBSTR('Delhi', 3, 2) → Output: lh.

## **Best Practices**

* Perform string trimming at the **program level** rather than the database side for better control and flexibility.

#### Additional Notes:

* Indexing for SQL string operations starts from **1**, unlike many programming languages where it starts from **0**.
* SQL functions like LOCATE, LEFT, RIGHT, and SUBSTR are useful for various string manipulations.

# Date-Related Functions

## **Getting Current Date and Time**

* NOW(): Returns the current date and time of the database server.
* Example:

SELECT NOW(); -- Output: 2024-12-31 12:52:38

* + Use case: Populating audit columns like created\_at or last\_modified\_at during inserts.

INSERT INTO STUDENTS VALUES (1, 'JOHN', NOW());

## **Fetching Current Date or Time Separately**

* CURRENT\_DATE: Returns only the current date.
* Example:

SELECT CURRENT\_DATE; -- Output: 2024-12-31

* CURRENT\_TIME: Returns only the current time.
* Example:

SELECT CURRENT\_TIME; -- Output: 12:56:29

## **Extracting Components from a Date**

* Use specific functions to extract parts of a date or time:
* YEAR(): Get the year from a date.

SELECT YEAR(NOW()); -- Output: 2024

* MONTH(): Get the month.

SELECT MONTH(NOW()); -- Output: 12

* DAY(): Get the day of the month.

SELECT DAY(NOW()); -- Output: 31

* HOUR(), MINUTE(),SECOND(): For time components.
* To get the name of the day (e.g., "Wednesday"), use DAYNAME().
* Example:

SELECT DAYNAME(NOW()); -- Output: Tuesday

## **Adding or Subtracting Intervals**

* DATE\_ADD(): Add an interval (e.g., days, hours, minutes) to a date.
* Example:

SELECT DATE\_ADD(NOW(), INTERVAL 7 DAY);

-- Adds 7 days to the current date. 2025-01-07 13:02:05

* Use case: Setting expiration times, such as locking seats for 10 minutes in a movie booking system:

UPDATE

    SEATS

SET

    LOCKED\_TILL = DATE\_ADD(NOW(), INTERVAL 10 MINUTE)

WHERE

    SEAT\_ID = 1;

## **Calculating the Difference Between Dates**

* DATEDIFF(): Calculates the difference in days between two dates.
* Example:

SELECT DATEDIFF('2023-03-01', '2023-02-15'); -- Output: 14

* Use case: Calculate the number of days taken to ship an order:

SELECT

    ORDER\_ID,

    DATEDIFF(SHIPPED\_DATE, ORDER\_DATE) AS DAYS\_TO\_SHIP

FROM

    ORDERS;

### **Practical Use Cases**

* **Audit Columns**: Automatically track creation and modification times.
* **Locking Mechanisms**: Use functions like DATE\_ADD() to manage timed locks (e.g., seats in booking systems).
* **Date Comparisons**: Use DATEDIFF() to calculate differences or determine overdue items.

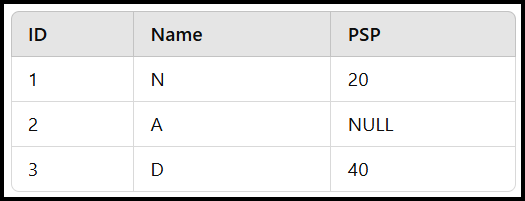
### **Key Notes**

* Always set a consistent time zone for your database to avoid confusion.
* For storing time zone specific data, ensure you use appropriate data types like TIMESTAMP WITH TIME ZONE.
* Testing and debugging SQL queries with date-related functions is crucial in systems requiring accurate time calculations.

# **IFNULL, COALESCE,** and **IF-ELSE**

## **Handling Null Values with** IFNULL**:**

* The IFNULL function allows you to replace NULL values with a specified value.
* Example:
* You have a STUDENTS table:



* You want to subtract 4 from PSP, but for NULL values, display "NOT AVAILABLE".

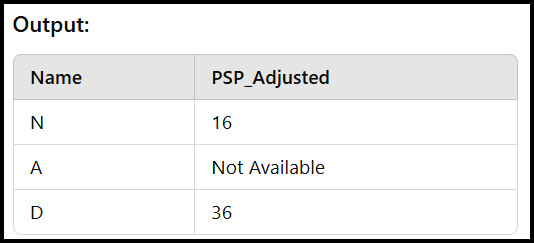
SELECT

    NAME,

    IFNULL(PSP - 4, 'NOT AVAILABLE') AS PSP\_ADJUSTED

FROM

    STUDENTS;



## **Handling Multiple Conditions with COALESCE:**

* The COALESCE function evaluates multiple values and returns the first non-NULL value.
* Example: You calculate a score based on PSP, assignments completed, or days spent in the course.
  + If PSP is available, use it.
  + Otherwise, use ASSIGNMENTS \* 2.
  + Otherwise, use (NUMBER OF DAYS / 5).
  + Otherwise, use 0.

SELECT

    NAME,

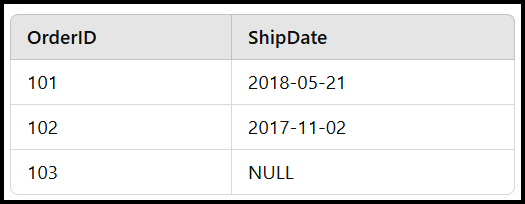
    COALESCE(PSP, ASSIGNMENTS \* 2, NUMBEROFDAYS / 5, 0) AS SCORE

FROM

    STUDENTS;

## **IF-ELSE:**

* To implement conditional logic, you can use the IF statement in SQL.
* Example: You want to print whether an order was shipped in 2018 or before.



SELECT

    ORDERID,

    IF(

        YEAR(SHIPDATE) = 2018,

        'SHIPPED IN 2018',

        'SHIPPED BEFORE 2018'

    ) AS SHIPMENTSTATUS

FROM

    ORDERS;

* Output:

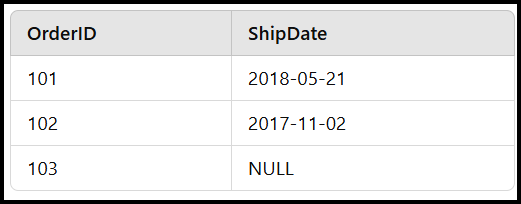


## **CASE statement**

* The CASE statement is a conditional expression that works like a switch or if-else structure in programming. It evaluates conditions sequentially and returns a result based on the first condition that evaluates to TRUE.

### **Example Use Case**

* We want to classify orders based on their shipping date:
  + If the shipping year is 2018, we want to label it as **"**SHIPPED IN 2018**"**.
  + Otherwise, we label it as **"**SHIPPED BEFORE 2018**"**.
  + If it is NULL, we label it as **"**NOT SHIPPED YET**"**.
* Here’s how to do it using CASE:



* Query:

SELECT

    ORDERID,

    CASE

        WHEN YEAR(SHIPDATE) = 2018 THEN 'SHIPPED IN 2018'

        WHEN YEAR(SHIPDATE) < 2018 THEN 'SHIPPED BEFORE 2018'

        ELSE 'NOT SHIPPED YET'

    END AS SHIPPINGSTATUS

FROM

    ORDERS;

* Explanation
  + CASE:
    - This starts the conditional block.
  + WHEN YEAR(SHIPDATE) = 2018:
    - This is the condition. If the year of SHIPDATE is 2018, it evaluates to TRUE.
  + THEN 'SHIPPED IN 2018':
    - If the condition in the WHEN clause is TRUE, it returns this value.
  + WHEN YEAR(SHIPDATE) < 2018:
    - If the condition in the WHEN clause is TRUE, it returns this value.
  + ELSE 'NOT SHIPPED YET'
    - If none of the WHEN conditions evaluate to TRUE, it falls to this default case.
  + END:
    - This marks the end of the CASE statement.
  + AS SHIPPINGSTATUS:
    - This is the alias for the calculated column.
* Output:

