Category	Function / Operator	Status	Description
Arithmetic Operators	`+`, `-`, `*`, `/`, `%`	✅ Covered	Basic math operations
Rounding	`CEIL()` / `CEILING()`	✅ Covered	Round up
Rounding	`FLOOR()`	✅ Covered	Round down
Rounding	`ROUND(x, d)`	✅ Covered	Round to d decimal places
Absolute Value	`ABS()`	✅ Covered	Return absolute value
Power & Root	`POWER(x, y)` / `POW(x, y)`	✅ Covered	x raised to y
Power & Root	`SQRT(x)`	✅ Covered	Square root
Exponential	`EXP(x)`	✅ Just Added	e^x
Logarithm	`LOG(x)`	✅ Covered	Natural log (ln)
Logarithm	`LOG10(x)`	✅ Covered	Log base 10
Trigonometric	`SIN(x)`, `COS(x)`, `TAN(x)`	✅ Covered	Trig functions
Trig Support	`RADIANS(x)`	✅ Covered	Degrees to radians
Trig Support	`DEGREES(x)`	✅ Just Added	Radians to degrees
Constants	`PI()`	✅ Just Added	Returns π
Sign-based	`SIGN(x)`	✅ Just Added	Returns -1, 0, or 1
Modular	`MOD(x, y)`	✅ Just Added	Remainder of division
Null-safe Arithmetic	`IFNULL(x, y)`	🔶 Not yet covered	Returns `y` if `x` is NULL
Null-safe Arithmetic	`COALESCE(x1, x2, ..., xn)`	🔶 Not yet covered	Returns first non-NULL value

1. Arithmetic Operators (`+`, `-`, `*`, `/`, `%`)

Scenario-Based Use Case:

Scenario: A shopkeeper wants to calculate the **total revenue from each product**, and also wants to find how much more or less a product costs compared to ₹3000.

```
SELECT
   product_name,
   price,
   quantity,
   price * quantity AS total_revenue,
   price - 3000 AS price_difference
FROM products;
```

When to Use:

- When calculating **totals**, **differences**, **tax**, **profit/loss**, etc.
- Any basic math operation required for analytics, reporting, or billing

2. `CEIL()`

✅ Scenario-Based Use Case:

Scenario: Courier service charges for **full kilograms only**, so a product weighing 1.2 kg is charged as 2 kg.

🔍 Query:

```
SELECT
    product_name,
    price,
    CEIL(price / 1000) AS shipping_blocks
FROM products;
```

📘 When to Use:

- When **rounding up** is required: shipping units, room bookings (minimum 1 hour), etc.
- Any scenario where partial units are charged fully

🔷 3. FLOOR()

✅ Scenario-Based Use Case:

Scenario: A discount is applied for every **₹1000** spent. So if a product costs ₹2950, the customer gets 2 full discount slabs.

🔍 Query:

```
SELECT
    product_name,
    price,
    FLOOR(price / 1000) AS discount_units
FROM products;
```

📘 When to Use:

- Round **down** to calculate bulk discount units, reward points
- When partial numbers should be ignored

🔷 4. ROUND()

✅ Scenario-Based Use Case:

Scenario: When displaying prices on an invoice, round them to 2 decimals for neat formatting.

🔍 Query:

```
SELECT
    product_name,
    ROUND(price, 2) AS rounded_price
FROM products;
```

📘 When to Use:

- Currency rounding in billing systems
- Display in dashboards or invoices

🔷 5. ABS()

✅ Scenario-Based Use Case:

Scenario: A manager wants to know **how far each product's price is from ₹3000**, regardless of whether it's higher or lower.

🔍 Query:

```
SELECT
    product_name,
    ABS(price - 3000) AS distance_from_3000
FROM products;
```

📘 When to Use:

- When calculating differences **without sign**
- Useful in **errors**, **deviations**, **price variance**

🔷 6. POWER(x, y)

✅ Scenario-Based Use Case:

Scenario: You want to calculate **compounded growth** (e.g., inflation or price projections over years).

🔍 Query:

```
SELECT
    product_name,
    POWER(price, 2) AS price_squared
FROM products;
```

📘 When to Use:

- Financial formulas like **compound interest**

- Scientific formulas (e.g., x^2, x^3)

🔷 7. SQRT()

✅ Scenario-Based Use Case:

Scenario: Engineer is analyzing the square root of price to normalize values before feeding to a machine learning model.

🔍 Query:

```
SELECT
    product_name,
    SQRT(price) AS root_price
FROM products;
```

📘 When to Use:

- Normalization
- Geometry, statistics, physics

🔷 8. LOG() and LOG10()

✅ Scenario-Based Use Case:

Scenario: Data analyst needs to compress large price values to analyze on a **logarithmic scale**.

🔍 Query:

```
SELECT
    product_name,
    LOG(price) AS natural_log,
    LOG10(price) AS base10_log
FROM products;
```

📘 When to Use:

- Scientific computations
- Economic modeling
- Data normalization (e.g., skewed price distributions)

🔷 9. Trigonometric Functions: `SIN()`, `COS()`, `TAN()`

✅ Scenario-Based Use Case:

Scenario: An engineer wants to calculate the **shadow projection angle** of a product's display board based on sun angle.

🔍 Query:

```
SELECT
    product_name,
    angle_degrees,
    SIN(RADIANS(angle_degrees)) AS sin_val,
    COS(RADIANS(angle_degrees)) AS cos_val,
    TAN(RADIANS(angle_degrees)) AS tan_val
FROM products;
```

📘 When to Use:

- Robotics
- Satellite or solar panel angle adjustments
- Gaming, physics simulation

se Case: Calculating Final Price with Default Value + Fixed Charge

Imagine you're managing an **online stationery store**.

- You have a table called `products` that stores:
    - `product_name`
    - `price`

- But **some products don't yet have a price** — maybe the new intern forgot to update it, so those fields are **NULL**.
- Now, you want to create a **temporary pricing report** to send to logistics that includes:
    - The current price (if available)
    - Or `₹0` if the price is missing
    - Plus a **fixed shipping/handling charge** of `₹100` for every item

```
SELECT
   product_name,
   price,
   IFNULL(price, 0) + 100 AS adjusted_price
FROM products;
```

`COALESCE(x1, x2, ..., xn)`

Description: Returns the first non-NULL value in the list.
Use Case: Fallback logic — if price is NULL, use discount_price; if that too is NULL, use 0.

```
SELECT
   product_name,
   COALESCE(price, discount_price, 0) AS effective_price
FROM products;
```

`MOD(a, b)`

Purpose: Returns remainder of a divided by b
Use Case: Check if a product ID is odd or even (e.g., for batch processing)

```
SELECT
    product_name,
    product_id,
    MOD(product_id, 2) AS is_even
FROM products;
```

📌 *When to Use*:

- For cyclic operations (e.g., alternate row coloring, batch scheduling)

- To identify divisibility

🔹 2. `EXP(x)`

Purpose: Returns e^x (e = 2.718…)
Use Case: For modeling exponential growth (e.g., price doubling every year)

```
SELECT
    product_name,
    EXP(2) AS growth_factor
FROM products;
```

📌 *When to Use*:

- Exponential growth or decay (e.g., virus spread, investments)

🔹 3. `DEGREES(x)`

Purpose: Converts radians to degrees
Use Case: Making trigonometric output user-readable

```
SELECT
    product_name,
    angle_degrees,
    RADIANS(angle_degrees) AS angle_radians,
    DEGREES(RADIANS(angle_degrees)) AS converted_back
FROM products;
```

📌 *When to Use*:

- Angle conversion in physics/engineering apps

🔹 4. `PI()`

Purpose: Returns π
Use Case: For circle-related area or circumference

```
SELECT
    PI() AS pi_value,
    PI() * POWER(5, 2) AS area_of_circle -- radius 5
```

📌 *When to Use*:

- Any geometric calculation involving circles

🔹 5. `SIGN(x)`

Purpose: Returns:

- `1` if x > 0
- `0` if x = 0
- `-1` if x < 0

Use Case: Find whether a price has increased or decreased compared to a benchmark

```
SELECT
    product_name,
    price,
    SIGN(price - 3000) AS price_trend
FROM products;
```

📌 *When to Use*:

- Detecting direction of change
- Categorizing data into increase, no change, or decrease

What is `TRUNCATE(number, decimal_places)` in MySQL?

The `TRUNCATE()` **function** is used to **cut off the decimal digits** to a specified number of decimal places — **without rounding**.

🔹 Syntax:

sql

CopyEdit

```
TRUNCATE(number, decimal_places)
```

- `number`: The numeric value you want to truncate.
- `decimal_places`: How many digits you want **after the decimal point**.
    - If it's positive → keeps that many decimal places.
    - If it's zero → returns an integer.
    - If negative → truncates digits **to the left of the decimal** (i.e., from the integer part).

🔹 Example Table: `samples`

id	value
1	12.7894
2	45.6789
3	9.9999
4	7.1

🔹 Query:

sql

CopyEdit

```
SELECT TRUNCATE(value, 2) FROM samples;
```

🔹 Output:

TRUNCATE(value, 2)
12.78
45.67
9.99
7.10

📌 The values are **cut off** after 2 decimal places. No rounding occurs.

🔁 Difference from `ROUND()`:

sql

CopyEdit

```
SELECT ROUND(12.7894, 2);   -- Output: 12.79
SELECT TRUNCATE(12.7894, 2);-- Output: 12.78
```

- `ROUND()` will **round up/down**.
- `TRUNCATE()` will **chop off** extra digits.

✅ Summary:

Function	Description	Example	Result
`TRUNCATE`	Removes digits without rounding	`TRUNCATE(5.6789, 2)`	`5.67`
`ROUND`	Rounds up or down	`ROUND(5.6789, 2)`	`5.68`