

## **Funciones definidas por el usuario (UDF's) – Common Tables Expressions (CTE)**

### **Funciones**

#### ¿Qué es una Función en MySQL?

Una función en MySQL es un bloque de código SQL que recibe cero o más argumentos, realiza una serie de operaciones y devuelve un único valor. Al igual que los procedimientos almacenados, las funciones permiten encapsular lógica y reutilizar código.

#### Tipos de Funciones en MySQL

##### **1. Escalares:**

- Devuelven un único valor y se pueden utilizar en cualquier lugar donde se permita una **expresión**.
- Ejemplo: calcular\_descuento (precio DECIMAL(7,2))

Cualquier combinación de valores, operadores y funciones que se evalúan para producir un valor

##### **2. Agregadas (algunas que ya conocemos):**

- Estas funciones realizan cálculos sobre un conjunto de valores y devuelven un valor único.
- Ejemplo: SUM, AVG, MAX, MIN.

### **Funciones Escalares**

Las funciones escalares operan sobre una sola fila de datos y devuelven un único valor. Son útiles cuando necesitamos realizar una operación sobre un valor individual en una consulta.

#### **Ejemplos**

##### **1. Funciones Matemáticas:**

- ABS(x): Devuelve el valor absoluto de x.
- ROUND(x, d): Redondea x a d decimales.
- SQRT(x): Devuelve la raíz cuadrada de x.

##### **2. Funciones de Cadena:**

- UPPER(str): Convierte una cadena a mayúsculas.
- LOWER(str): Convierte una cadena a minúsculas.
- CONCAT(str1, str2, ...): Concatenar varias cadenas.

##### **3. Funciones de Fecha:**

- NOW(): Devuelve la fecha y hora actuales.
- DATE\_ADD(date, INTERVAL expr type): Añade un intervalo a una fecha.
- YEAR(date): Devuelve el año de una fecha.

### Funciones Agregadas

Las funciones agregadas operan sobre un conjunto de filas y devuelven un único valor que resume la información de ese conjunto. Son útiles cuando necesitamos calcular estadísticas o resúmenes de datos.

### Ejemplos de Funciones Agregadas

- Max
- Min
- Sum
- Avg
- Count

### Tabla Comparativa entre tipos de funciones

Característica	Funciones Escalares	Funciones Agregadas
Operación	Sobre una sola fila	Sobre un conjunto de filas
Resultado	Un único valor	Un único valor que resume el conjunto
Uso	Cálculos, transformaciones y operaciones	Resúmenes, estadísticas y agregaciones
Ejemplos	ABS, ROUND, UPPER, NOW	SUM, AVG, COUNT, MAX, MIN
Aplicación	SELECT ABS(columna) FROM tabla	SELECT SUM(columna) FROM tabla

### Sintaxis Básica de una Función

```
CREATE FUNCTION nombre_funcion(parametro1 tipo_dato, parametro2 tipo_dato, ...)
RETURNS tipo_dato_retorno
BEGIN
    /* aquí escribimos el código de la función */
    RETURN valor_de_retorno;
END;
```

Ejemplo: Función para Calcular el Descuento

```
CREATE FUNCTION calcular_descuento(precio DECIMAL(7,2))
DETERMINISTIC
RETURNS DECIMAL(7,2)
BEGIN
    DECLARE descuento DECIMAL(7,2);
    SET descuento = precio * 0.1;
    RETURN precio - descuento;
END;
```

Indica que la función devuelve el mismo resultado para los mismos parámetros de entrada.

**Cuidado!!!**  
(no es lo mismo)

## Funciones Determinísticas

Una función se considera **determinística** si devuelve el **mismo resultado** cada vez que se invoca con los mismos parámetros (no necesariamente con los mismos valores). Esto significa que su salida es predecible y constante para los mismos valores de entrada. Ejemplos típicos son `ABS()` o `LENGTH()`.

## Funciones No Determinísticas

Por otro lado, una función se considera **no determinística** si su salida puede variar aunque se invoque con los mismos parámetros. Esto puede suceder si la función depende de variables externas, datos en tablas que pueden cambiar, o funciones como `NOW()`, que devuelven la fecha y hora actual.

### Importancia del Determinismo:

- **Optimización:** Saber que una función es determinística permite al motor de base de datos optimizar consultas y reutilizar resultados.
- **Consistencia:** Garantiza resultados consistentes y predecibles para las mismas entradas.
- **Caché:** Funciones determinísticas pueden ser evaluadas una vez y almacenadas en caché para mejorar el rendimiento.
- **Replicación y Restauración:** Especificar el determinismo ayuda en la replicación de bases de datos. MySQL necesita saber si los resultados de las funciones son predecibles para mantener la consistencia entre el servidor maestro y los servidores esclavos.
- **Documentación y Mantenimiento:** Indicar si una función es determinística o no ayuda a otros desarrolladores a comprender mejor cómo funciona la función y qué esperar de ella, lo que facilita el mantenimiento del código.

### Por otro lado...

Si no especificamos `DETERMINISTIC` o `NOT DETERMINISTIC`, MySQL asumirá por defecto que la función es **no determinística**. Esto puede llevar a menos optimizaciones y afectar el rendimiento en ciertos casos.

Por lo tanto, aunque no es obligatorio, es una buena práctica especificar si una función es determinística o no para aprovechar al máximo las capacidades de optimización y mantener la claridad del código.

## Comparación entre Funciones y Procedimientos Almacenados en MySQL

### 1. Funciones

- Retorno de un valor: El objetivo principal de una función es realizar un cálculo y devolver un único valor.
- Uso en expresiones: Pueden ser utilizadas directamente dentro de expresiones SQL, como en el `SELECT`, `WHERE` o `HAVING`.
- Sintaxis: Tienen una sintaxis específica que incluye la cláusula `RETURNS` para indicar el tipo de dato del valor a devolver.
- Limitaciones: Generalmente no realizan operaciones de modificación de datos (`INSERT`, `UPDATE`, `DELETE`) de forma directa.
- Ejemplos de uso:
  - o Calcular el precio total de una venta.
  - o Obtener el nombre completo de un cliente.
  - o Verificar si un valor existe en una tabla.

## 2. Procedimientos Almacenados (SP)

- Conjunto de instrucciones: Son bloques de código SQL que pueden realizar múltiples operaciones, incluyendo consultas, actualizaciones y llamadas a otras funciones o procedimientos.
- No retornan un valor: Aunque pueden modificar datos, no devuelven un valor directamente. Para obtener resultados, se utilizan variables de salida o se generan conjuntos de resultados.
- Uso: Se invocan mediante la sentencia CALL.
- Flexibilidad: Ofrecen mayor flexibilidad al permitir realizar operaciones más complejas y controlar el flujo de ejecución del código.
- Ejemplos de uso:
  - o Insertar nuevos registros en múltiples tablas.
  - o Actualizar varios registros de una tabla.
  - o Realizar transacciones complejas.

### **Tabla comparativa entre Funciones y SP**

Característica	Funciones	Procedimientos Almacenados
Propósito principal	Calcular y devolver un valor	Realizar múltiples operaciones
Uso en expresiones	Sí	No
Retorno de valor	Sí, uno solo	No (en el sentido estricto), pero pueden modificar datos
Sintaxis	CREATE FUNCTION	CREATE PROCEDURE
Invocación	En expresiones SQL	CALL
Complejidad	Generalmente más simple	Pueden ser más complejos

### **Cuando utilizar una función**

- **Cálculos simples:** Cuando necesitamos realizar operaciones matemáticas o manipulaciones de datos relativamente sencillas, una función es ideal. Por ejemplo, calcular el descuento de un producto, obtener la longitud de una cadena de texto o convertir una fecha a un formato específico.
- **Validaciones:** Las funciones son útiles para validar datos de entrada, como verificar si un valor está dentro de un rango específico o si un formato es correcto.
- **Reutilización en expresiones:** Podemos utilizar funciones directamente en las consultas SQL, lo que facilita la reutilización de código y mejora la legibilidad.
- **Retorno de un único valor:** Si necesitamos obtener un solo resultado como salida, una función es la opción más directa.

### **Cuando utilizar un procedimiento almacenado**

- **Operaciones complejas:** Cuando requerimos realizar múltiples operaciones, como insertar, actualizar o eliminar datos en varias tablas, un SP es más adecuado.
- **Transacciones:** Los SP son ideales para gestionar transacciones, asegurando que un conjunto de operaciones se ejecute como una unidad atómica.
- **Procedimientos de negocio:** Si tenemos una serie de pasos bien definidos que conforman un proceso de negocio, un SP puede encapsular toda esa lógica.
- **Control de flujo:** Los SP nos permiten utilizar estructuras de control como IF, ELSE, WHILE y FOR para tomar decisiones y repetir bloques de código.
- **Parámetros de entrada y salida:** Los SP pueden aceptar múltiples parámetros de entrada y devolver resultados a través de parámetros de salida o conjuntos de resultados.

#### **Ejemplo:**

Supongamos que tenemos una tienda en línea y necesitamos calcular el precio final de un pedido, aplicando descuentos y calculando impuestos.

- **Función:** Podemos crear una función `calcular_precio_final` que tome como entrada el precio base, el descuento y el impuesto, y devuelva el precio final.
- **Procedimiento almacenado:** Podemos crear un SP que calcule el precio final, actualice la tabla de pedidos y envíe un correo electrónico de confirmación.

### ***Función para calcular el precio final de una venta***

```
CREATE FUNCTION calcular_precio_final(venta_id INT, descuento DECIMAL(7,2),  
impuesto DECIMAL(7,2))  
RETURNS DECIMAL(10,2)  
BEGIN  
    DECLARE subtotal DECIMAL(10,2);  
    DECLARE total DECIMAL(10,2);  
    SELECT precio * cantidad INTO subtotal  
    FROM ventas  
    WHERE venta_id = venta_id;  
    SET total = subtotal * (1 - descuento) * (1 + impuesto);  
    RETURN total;  
END;
```

***Procedimiento almacenado para actualizar el precio final en la tabla ventas***

```
CREATE PROCEDURE actualizar_precio_final(venta_id INT, descuento DECIMAL(7,2),  
impuesto DECIMAL(7,2))  
BEGIN  
    UPDATE ventas  
    SET precio = calcular_precio_final(venta_id, descuento, impuesto)  
    WHERE venta_id = venta_id;  
END;
```

Llamado al SP:

CALL actualizar\_precio\_final(10, 0.1, 0.21); -- Actualizamos la venta con ID 10, con un descuento del 10% y un impuesto del 21%

**Ejercicios de Funciones (Consignas)**

Ejercicio 1: Obtener el Nombre Completo de un Cliente

Ejercicio 2: Obtener el Total Vendido de un Libro

Ejercicio 3: Obtener el Cliente que Más Ha Gastado

Ejercicio 4: Obtener el Libro Más Vendido en un Año

Ejercicio 5: Verificar si un Cliente Comprado un Libro en Particular

## **Introducción a las CTE (Common Table Expressions)**

Una CTE (Expresión de Tabla Común) es una forma temporal de definir una tabla derivada en una consulta SQL. También podemos decir que son una característica de SQL que permite crear una "tabla temporal" que puede ser referenciada dentro de una consulta.

A diferencia de las subconsultas, las CTE son más legibles y reutilizables. Su uso se recomienda especialmente cuando una consulta tiene múltiples partes complejas o cuando se necesita hacer referencia a un mismo conjunto de datos varias veces dentro de una consulta (CTE recursiva).

### **Tipos de CTE**

#### **1. CTE Simples:**

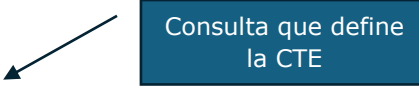
- Una CTE simple se define usando la cláusula WITH seguida por una consulta SQL.
- La tabla temporal creada por la CTE solo está disponible durante la ejecución de la consulta principal que la sigue.

#### **2. CTE Recursivas:**


- Las CTE recursivas permiten hacer referencia a sí mismas, facilitando el trabajo con datos jerárquicos o con estructuras recursivas, como árboles genealógicos, estructuras organizativas, o datos que requieren cálculos iterativos.

### **Sintaxis básica de una CTE (simple):**

```
WITH mi_CTE AS  
(  
    SELECT columnas  
    FROM tabla  
    WHERE condiciones  
)
```



```
Consulta principal que utiliza la CTE  
  
SELECT columnas  
FROM mi_CTE;
```



- WITH: Palabra clave que introduce la CTE.
- mi\_CTE: Nombre temporal de la CTE.
- La consulta después de WITH define los datos que estarán disponibles dentro de la CTE.
- La consulta principal puede usar la CTE como si fuera una tabla temporal.

### **Sintaxis de una CTE recursiva:**

```
WITH RECURSIVE nombre_cte_recursiva AS  
(  
    -- Este seria el "punto de partida" de la recursión  
    SELECT columna1, columna2, ...  
    FROM tabla  
    WHERE condición_inicial  
  
    UNION ALL  
  
    -- Aquí podemos ver la recursividad (se hace referencia a la CTE)  
    SELECT columna1, columna2, ...  
    FROM tabla  
    JOIN nombre_cte_recursiva ON condición_recursiva  
)
```

Consulta principal que utiliza la CTE recursiva

```
SELECT columna1, columna2, ...  
FROM nombre_cte_rekursiva;
```

---

**Ventajas de las CTE:**

1. Legibilidad: Hace que las consultas sean más fáciles de leer y entender, especialmente cuando se tiene que trabajar con múltiples subconsultas.
2. Reutilización: Las CTE permiten usar el mismo conjunto de datos en diferentes partes de la consulta sin tener que escribir la subconsulta repetidamente.
3. Recursividad: Las CTE también pueden ser recursivas, lo que permite resolver problemas como el manejo de jerarquías o la iteración a través de conjuntos de datos.

**Ejemplos de CTE**

Ejemplo 1: CTE básica para calcular promedios

```
WITH EmpleadosSalarios AS  
(  
    SELECT empleado_id, salario  
    FROM Empleados  
    WHERE salario > 3000  
)
```

```
SELECT AVG(salario) AS PromedioSalario  
FROM EmpleadosSalarios;
```

la CTE EmpleadosSalarios  
selecciona los empleados con  
un salario superior a 3000

la consulta principal calcula  
el salario promedio de esos  
empleados

Ejemplo 2: CTE con JOIN

```
WITH EmpleadosDepartamentos AS  
(  
    SELECT e.empleado_id, e.nombre, d.nombre_departamento  
    FROM Empleados e  
    JOIN Departamentos d ON e.departamento_id = d.departamento_id  
)
```

```
SELECT *  
FROM EmpleadosDepartamentos  
ORDER BY nombre_departamento;
```



Ejemplo 3: CTE recursiva

```
WITH RECURSIVE jerarquia_empleados AS  
(  
    /*obtener el jefe inicial*/  
  
    SELECT id, nombre, id_jefe  
    FROM empleados  
    WHERE id_jefe IS NULL /*Comenzamos con el jefe máximo*/  
  
    UNION ALL  
  
    /*Parte recursiva: obtener los empleados supervisados por los empleados obtenidos en la  
    parte inicial*/  
  
    SELECT e.id, e.nombre, e.id_jefe  
    FROM empleados e  
    JOIN jerarquia_empleados je ON e.id_jefe = je.id  
)  
  
/*consulta ppal (llamado a la CTE recursiva)*/  
  
SELECT *  
FROM jerarquia_empleados;
```

la CTE recursiva  
Jerarquía\_empleados se  
utiliza para construir una  
jerarquía de  
empleados y sus jefes

### Diferencias entre SubConsultas y CTE's (tabla comparativa)

A continuación, tenemos una tabla comparativa en la que podemos observar las características principales que comparten o no, las subqueries y las cte's.

Característica	Subconsultas	CTE (Common Table Expressions)
<b>Definición</b>	Consultas anidadas dentro de una consulta principal	Bloques de consulta temporales referenciables
<b>Sintaxis</b>	Parte de las cláusulas SELECT, FROM, WHERE, etc.	Definida con WITH y puede ser recursiva
<b>Legibilidad</b>	Puede volverse compleja y difícil de leer	Mejora la legibilidad dividiendo consultas complejas
<b>Reutilización</b>	No puede ser reutilizada dentro de la misma consulta	Puede ser referenciada múltiples veces en una consulta
<b>Recursividad</b>	No admite recursividad	Admite recursividad (útil para datos jerárquicos)
<b>Compatibilidad</b>	Compatibles con casi todas las versiones de SQL	Requiere versiones posteriores a MySQL 8.0

**Ejemplo (para comparar)**

1. *Subconsulta para Obtener el Cliente con más facturación*

```
SELECT cliente_id, nombre
FROM clientes
WHERE cliente_id = (SELECT cliente_id
                    FROM ventas
                    GROUP BY cliente_id
                    ORDER BY SUM(total) DESC
                    LIMIT 1);
```

2. *CTE para Obtener el Cliente con más facturación*

```
WITH ventas_totales AS
(
  SELECT cliente_id, SUM(total) AS total_ventas
  FROM ventas
  GROUP BY cliente_id
)

SELECT c.cliente_id, c.nombre
FROM clientes c
JOIN ventas_totales v ON c.cliente_id = v.cliente_id
ORDER BY v.total_ventas DESC
LIMIT 1;
```

**Entonces, por lo visto hasta aquí, ¿cuándo nos conviene usar una SubQuery y cuando una CTE?**

**CTE cuando:**

- Queremos mejorar la legibilidad y estructura del código.
- Necesitamos reutilizar resultados intermedios.
- Trabajamos con datos jerárquicos o recursivos.
- Deseamos separar la lógica de negocio en componentes claros.
- Utilizamos versiones modernas de MySQL (8.0 o superior).

**Subconsultas cuando:**

- La consulta sea simple y no necesite ser reutilizada en múltiples lugares.
- No tengamos que trabajar con datos recursivos.
- La compatibilidad con versiones más antiguas de MySQL sea un factor.