CONSULTAS AVANZADAS II

FUNCIONES

Diseño y Programación de una BBDD

ÍNDICE

CONSULTAS AVANZADAS II. FUNCIONES

1. FUNCIONES CONDICIONALES.

a. Explicar para qué y cómo (poner un ejemplo) se utilizan las siguientes funciones condicionales de MySQL: CASE, COALESCE, IF, IFNULL, NULLIF. Indica las diferencias entre ellas.

CASE: La función condicional “CASE” se utiliza para realizar evaluaciones condicionales y devolver valores diferentes según se cumplan ciertas condiciones. Puede ser usada de dos maneras, de forma simple o de forma con múltiples condiciones.

CASE Simple, se utiliza para realizar una evaluación y devolver un valor si se cumplen una condición determinada.

En el siguiente ejemplo se toma como referencia un taller mecánico.

```
SELECT
    id_servicio,
    descripcion,
    costo,
    CASE
        WHEN tipo_servicio = 'cambio_aceite' OR tipo_servicio = 'revision_frenos' THEN 'Mantenimiento'
        WHEN tipo_servicio = 'reparacion_motor' OR tipo_servicio = 'cambio_transmision' THEN 'Reparación'
        WHEN tipo_servicio = 'inspeccion_general' THEN 'Inspección'
        ELSE 'Otro'
```

```
    END AS categoria_servicio
FROM
    servicios;
```

Por otro lado, se puede utilizar “CASE” con múltiples condiciones en una sola expresión.

En el siguiente ejemplo la función “CASE” se utiliza para clasificar los servicios según su tipo (‘tipo_servicio’) y su coste.

```
SELECT
    id_servicio,
    descripcion,
    costo,
CASE
WHEN tipo_servicio = 'cambio_aceite' THEN
CASE
WHEN costo < 50 THEN 'Bajo costo - Mantenimiento básico'
WHEN costo >= 50 AND costo < 100 THEN 'Costo moderado - Mantenimiento básico'
ELSE 'Alto costo - Mantenimiento básico'
END
WHEN tipo_servicio = 'reparacion_motor' THEN
```

```
CASE
WHEN costo < 200 THEN 'Bajo costo - Reparación'
WHEN costo >= 200 AND costo < 500 THEN 'Costo moderado - Reparación'
ELSE 'Alto costo - Reparación'
END
WHEN tipo_servicio = 'inspeccion_general' THEN
CASE
WHEN costo < 100 THEN 'Bajo costo - Inspección general'
WHEN costo >= 100 AND costo < 300 THEN 'Costo moderado - Inspección general'
ELSE 'Alto costo - Inspección general'
END
ELSE 'Otro tipo de servicio'
END AS categoria_costo
FROM
    Servicios;
```

COALESCE: La función “COALESCE” se utiliza para retornar el primer valor no nulo de una lista de expresiones. Si alguna de las expresiones es no nula, devuelve ese valor. De lo contrario devuelve NULL si todas son nulas.

En el siguiente ejemplo para mostrar la fecha de la última revisión o un mensaje indicando que la revisión aún no se ha realizado:

```
SELECT
    id_vehiculo,
    marca,
    modelo,
COALESCE(fecha_revision, 'Revisión pendiente') AS estado_revision
FROM
    vehiculos;
```

IF: La función condicional “IF” se utiliza para realizar evaluaciones lógicas y devolver un valor en función de si una condición es verdadera o falsa.

En el siguiente ejemplo realizamos la condición si la duración real del servicio fue menor o igual a la estimada.

```
SELECT
    id_servicio,
    descripcion,
    duracion_estimada,
    duracion_real,

    IF(duracion_real <= duracion_estimada, 'A tiempo', Con demora) AS estado_servicio
FROM
    servicios;
```

IFNULL: La función condicional “IFNULL” se utiliza para tratar valores nulos en las consultas y devolver un valor alternativo si es valor es null.

En el siguiente ejemplo indicamos una condiciona IFNULL para mostrar una fecha alternativa si la fecha de la última revisión es null.

```
SELECT
    id_vehiculo,
    marca,
    modelo,
    IFNULL(fecha_ultima_revision, 'Sin revisión') AS estado_revision
FROM
    vehiculos;
```

NULLIF: La función “NULLIF” se utiliza para comparar dos expresiones y devolver NULL si son iguales o devolver la primera expresión si son diferentes.

En el siguiente ejemple se comparar el nombre del propietario con el nombre del conductor. Si los nombres son diferentes mostrará el nombre del propietario en dicha columna.

```
SELECT
    id_vehiculo,
    nombre_propietario,
    nombre_conductor,
    NULLIF(nombre_propietario, nombre_conductor) AS
diferencia_propietario_conductor
```

```
FROM
    vehiculos;
```

Cada función tiene su propio propósito y son distintas entre sí:

- **CASE:** Es flexible para múltiples condiciones.
- **COALESCE:** Reemplaza valores nulos por uno alternativo.
- **IF:** Evaluación simple de una condición.
- **IFNULL:** Reemplaza un valor null por otro especificado.
- **NULLIF:** Devuelve null si dos expresiones son iguales, sino devuelve la primera expresión.

2. FUNCIONES UDF (FUNCIONES ALMACENADAS).

a. ¿Qué es una función almacenada o función definida por el usuario (UDF)?

Las funciones almacenadas son un conjunto de instrucciones SQL que se agrupan y almacenas en el servidor de base de datos para ser utilizadas repetidamente.

Entre sus principales características tiene la reutilización de código permitiendo reutilizar lógica en múltiples consultas o procedimientos almacenados dentro de la base de datos. Ayuda a dividir la lógica del negocio en funciones más pequeñas, lo que facilita la comprensión del código y el mantenimiento.

Mejoran el rendimiento al reducir la duplicación de código y al permitir la ejecución de operaciones complejas en el servidor en lugar de realizarlo en la aplicación cliente.

Por último, admiten varios tipos de funciones almacenadas, como funciones definidas por usuario o funciones de procedimiento almacenados y funciones definidas por el usuario con parámetros.

b. Mostrar la sintaxis básica de una función UDF y para qué se utiliza cada instrucción.

```
CREATE FUNCTION calcular_coste_total(coste_servicio DECIMAL(10, 2))

RETURNS DECIMAL(10, 2)

BEGIN

    DECLARE impuesto DECIMAL(10, 2);

    DECLARE coste_total DECIMAL(10, 2);



    SET impuesto = coste_servicio * 0.21; -- Calcula el impuesto como el 21% del coste
    del servicio

    SET coste_total = coste_servicio + impuesto; -- Calcula el coste total sumando el
    impuesto al coste del servicio

    RETURN coste_total; -- Devuelve el coste total, incluyendo el impuesto

END;
```

CREATE FUNTION: se utiliza para crear una función UDF , en el ejemplo "calcular_coste_total".

RETURNS: indic que la función devolverá un valor, en nuestro ejemplo un valor decimal (10,2).

BEGIN y END: delimitan el bloque de código de la función.

DECLARE: se utiliza para declarar variables locales dentro de la función.

RETURN: devuelve el valor, en nuestro caso el coste total.

c. ¿Cómo se utiliza una función UDF?

Para utilizar la función utilizaremos una llamada dentro de una consulta SQL.

SELECT nombre_funcion(argumento1, argumento2, …)

FROM tabla;

En nuestro ejemplo anterior sería el siguiente:

SELECT calcular_coste_total(1000) AS coste_total_servicio;

3. VARIABLES DE @usuario.

a. ¿Qué es una variable de usuario?

Son variables definidas por el usuario que permiten almacenar valores temporalmente durante la sesión de conexión. Estas variables pueden ser utilizadas para almacenar información que puede ser referenciada y modificada dentro de la consulta MySQL.

En cuanto a sus características se encuentra que las variables de usuario tienen un ámbito limitado a la sesión de conexión actual. Una vez cerrada la sesión se cierra y las variables se pierden.

b. ¿Cómo se define una variable de usuario? ¿Qué requisitos debe tener su nombre?

Para definir una variable de usuario utilizamos el símbolo @ **s**eguido por el nombre que le asignemos a la variable. Para asignar un valor a la variable, podemos utilizar el operador **:=** o la función **SET**.

SET @ejemplo_variable := 'Valor_variable';

c. ¿Cómo se utilizan las variables de usuario dentro de una instrucción SELECT?

Para utilizar una variable dentro de una instrucciones SELECT lo realizamos de la siguiente manera:

SELECT * FROM table WHERE columna = @ejemplo_variable;

d. ¿Cómo guardar valores de una consulta en una variable de usuario?

Para guardar valores de una consulta en una variable de usuario de usuario realizaremos los siguientes pasos:

Primero declararemos e inicializaremos la variable:

SET @total_registros = 0;

Procedemos a realizar la consulta para obtener el número de registros:

SELECT COUNT(*) INTO @total_registros FROM mi_tabla;

Mostramos el valor almacenado en la variable:

SELECT @total_registros AS total_registros;