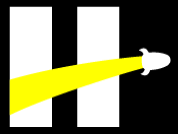
Variables, funciones y procedimientos



Variables

En los scripts SQL, se pueden utilizar variables para almacenar valores durante la ejecución de una secuencia de comandos y utilizarlos luego.

MySQL y SQL Server entre otros, reconocen diferentes tipos de variables.

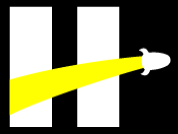
Entre ellas podemos encontrar: definidas por el usuario, locales y del sistema.

Definidas por el usuario

Identificadas por un símbolo @. Para inicializar una variable definida por el usuario, necesitas usar una declaración SET. Se puede inicializar muchas variables a la vez, separando cada declaración de asignación con una coma. Una vez que asignas un valor a una variable, tendrá un tipo de acuerdo al valor dado.

```
SET @carrera1 = 'Data Science',@carrera2 = 'Full Stack'

SELECT @carrera1, @carrera2
```

```
SELECT @cod := codigo FROM cohortes WHERE idCohorte = 1235;

SELECT @cod

SELECT fechaInicio
FROM cohortes
WHERE codigo = @cod
```

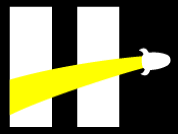


Locales

Este tipo de variables no necesitan el prefijo @ en sus nombres, se declararan antes de que puedan ser usadas. Se pueden utilizar variables locales de dos maneras:

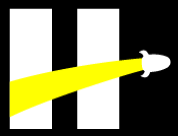
- Utilizando la declaración DECLARE.
- Como un parámetro dentro de una declaración STORED PROCEDURE. Se utilizan variables en los procedimientos almacenados para obtener resultados inmediatos. Estas variables son locales dentro del procedimiento.

Cuando se declara una variable local, se suele asignar un valor por defecto. La variable se inicializa con un valor NULL si no asignas ningún valor.



Locales

```sql
-- Más abajo daremos más detalles sobre los procedimientos almacenados.
DELIMITER $$

CREATE PROCEDURE GetTotalAlumnos()
BEGIN
        DECLARE totalAlumnos INT DEFAULT 0;

    SELECT COUNT(*)
    INTO totalAlumnos
    FROM alumnos;

    SELECT totalAlumnos;
END$$

DELIMITER ;

CALL GetTotalAlumnos()
```



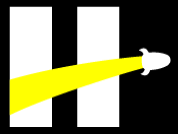
Variables del Sistema

Las variables del sistema se identifican con un doble signo @ o utilizando las palabras GLOBAL o SESSION en la sentencia SET.

Indican la configuración por defecto y pueden ser modificadas en caso de ser necesario.

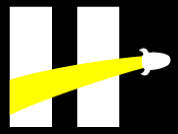
Para ver las variables de sistema en uso dentro de una sesión o en el servidor, se utiliza la sentencia SHOW VARIABLES.

Es necesario a veces conocer el valor de estas variables, como el caso de la versión que se está utilizando, o conocer y cambiar un timeout. Son muy útiles para quienes tratan de mejorar el rendimiento de su sesión.



Variables del Sistema

```
SHOW VARIABLES -- Muestra todas las variables.

SHOW SESSION VARIABLES
SHOW LOCAL VARIABLES
SHOW VARIABLES

-- Se puede utilizar el operador LIKE '%Variable%' para acceder a una variable en particular.
-- Ejemplo:

SHOW SESSION VARIABLES LIKE 'version'

SHOW SESSION VARIABLES LIKE 'version_comment'
```



Funciones

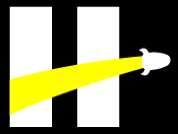
Las funciones nos permiten procesar y manipular datos de un modo muy eficiente.

Existen funciones integradas dentro de SQL, algunas de las que utilizamos son AVG, SUM, CONCATENATE, etc.

Pueden ser utilizadas en las sentencias SQL independientemente del lenguaje de programación del servidor sobre el que se ejecuten las consultas. Las funciones también se pueden crear dentro de SQL y esto permite personalizar ciertas operaciones propias del proyecto. Para poder crear funciones se deben tener los permisos INSERT y DELETE.

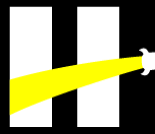
Funciones

```sql
-- EJEMPLO 1:

-- Esta función recibe una fecha de ingreso y cálcula la antigüedad en meses del alumno.

DELIMITER $$
CREATE FUNCTION antiguedadMeses(fechaIngreso DATE) RETURNS INT -- Asignamos un nombre, parámetros de la función y tipo de dato a reto

-- La función se define entre BEGIN - END.
BEGIN
        DECLARE meses INT; -- Declaramos las variables que van a operar en la función
        SET meses = TIMESTAMPDIFF(MONTH, fechaIngreso, DATE(NOW())); - -- Definimos el script.
    RETURN meses; -- Retornamos el valor de salida que debe coincidir con el tipo declarado en CREATE
END$$

DELIMITER ;

SELECT * , antiguedadMeses(fechaIngreso)

FROM alumnos
```



Funciones

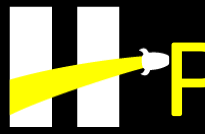
```sql
-- EJEMPLO 2:

-- Esta función recibe el id de un alumno y devuelve su antigüedad en meses.

DELIMITER $$
CREATE FUNCTION antiguedadMeses2(id INT) RETURNS INT
BEGIN
        DECLARE meses INT;
    SELECT TIMESTAMPDIFF(MONTH, fechaIngreso, DATE(NOW()))
    INTO meses
    FROM alumnos
    WHERE idAlumno = id;
    RETURN meses;
END$$

DELIMITER ;

SELECT antiguedadMeses2(130)
```



Procedimientos Almacenados

Es un objeto que se crea con la sentencia CREATE PROCEDURE y se invoca con la sentencia CALL. Un procedimiento puede tener cero o muchos parámetros de entrada y cero o muchos parámetros de salida.

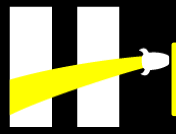
Resulta útil cuando se repite la misma tarea repetidas veces, siendo un buen método para encapsular el código. Al igual que ocurre con las funciones, también puede aceptar datos como parámetros, de modo que actúa en base a éstos.



Procedimientos Almacenados

Los parámetros de los procedimientos almacenados de MySQL pueden ser de tres tipos:

- **IN:** Es el tipo de parámetro que se usa por defecto. La aplicación o código que invoque al procedimiento tendrá que pasar un argumento para este parámetro.
- **OUT:** El valor de este parámetros pude ser cambiado en el procedimiento, y además su valor modificado será enviado de vuelta al código.
- INOUT: Es una mezcla de los dos conceptos anteriores. La aplicación o código que invoca al procedimiento puede pasarle un valor a éste, devolviendo el valor modificado al terminar la ejecución.



Procedimientos Almacenados

```
-- Este procedimiento lista los alumnos pertenecientes a una carrera.

DELIMITER $$
CREATE PROCEDURE listarCarrera( IN nombreCarrera VARCHAR(25))
BEGIN
        SELECT CONCAT(alumnos.nombre,' ',apellido) AS Alumno, cohorte
        FROM alumnos
        INNER JOIN cohortes
        ON cohorte = idCohorte
        INNER JOIN carreras
        ON carrera = idCarrera
        WHERE carreras.nombre=nombreCarrera;
END;

DELIMITER

CALL listarCarrera('Data Science')
```