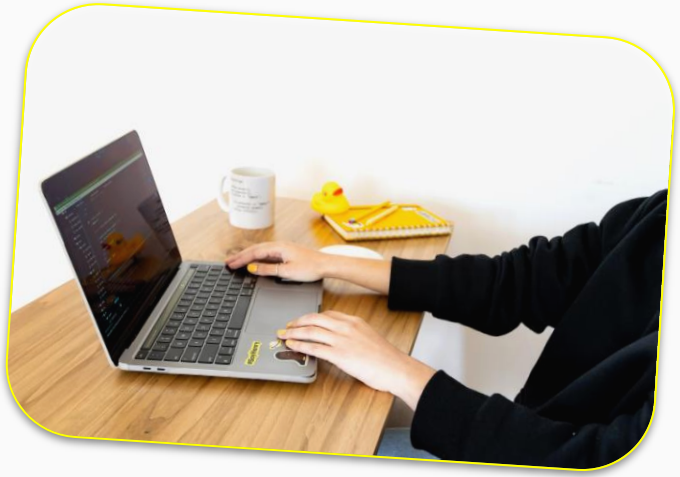
HENRY

Variables, funciones, y procedimientos

Data Science

Agenda



- Tipos de variables: definidas por el usuario, locales, funciones,
- Procedimientos Almacenados

OBJETIVOS DE LA CLASE

Al finalizar esta lecture estarás en la capacidad de...

- → **Aplicar** las variables, funciones y procedimientos en SQL

Variables

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

Definidas por el usuario

Identificadas por un **símbolo @.** Para inicializar una variable definida por el usuario, debes de 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.

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

SELECT @cod

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

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

SELECT @carrera1, @carrera2
```

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

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

SELECT @cod

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

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

SELECT @carrera1, @carrera2
```

Locales

Este tipo de variables no necesitan el **prefijo @** en sus nombres, se declaran 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**.

La variable se inicializa con un **valor NULL** si no asignamos ningún valor.

```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.

En ocasiones es necesario conocer el **valor** de estas variables, como el caso de la versión que se está utilizando, o conocer y cambiar un **timeout**.

```
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**.

```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
```

```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

¿Qué son?

se crea con la sentencia **CREATE PROCEDURE**

se invoca con la sentencia **CALL**

Un procedimiento puede tener cero o muchos parámetros de entrada y cero o muchos parámetros de salida.

Acepta **datos** como **parámetros**, actúa con base a éstos.

Pueden ser de tres tipos

01 **IN**: se usa por defecto. La aplicación o código que invoque al procedimiento tendrá que pasar un argumento para este parámetro.

02 **OUT**: el valor de este parámetros puede ser cambiado en el procedimiento, y su valor modificado será enviado de vuelta al código.

03 **INOUT**: mezcla de los dos conceptos anteriores. La aplicación o código que invoca al procedimiento puede pasarle un valor a éste, devuelve el valor modificado al terminar la ejecución.

```sql
-- 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')
```


HENRY