CLASE 16

*STORED PROCEDURES*

*CONCEPTO GENERAL*

Un Stored Procedure o **Procedimiento Almacenado** representa un conjunto de sentencias almacenado físicamente en una DB, creado para cumplir tareas específicas.

Permite también establecer niveles de seguridad y manipular operaciones complejas o extensas del lado del servidor, evitando un ida y vuelta de datos que termine sobrecargando una red o servidor.Su estructura es similar a las **Funciones SQL** que vimos en la clase anterior pero, a diferencia de éstas, en un **Stored Procedure** su objetivo es resolver desde una operación simple hasta operaciones complejas que requieran modificar varias tablas y/o datos almacenados en una DB.

El Lenguaje de programación **SQL** es el lenguaje usado para crear su lógica e integra también muchos comandos del tipo condicional, variables de entrada y de salida, potenciando así su poder de acción.

En el manejo de procesos complejos los S.P. Se usan como una especie de API que ejecuta consultas, compara resultados, actualiza datos en algunas tablas y/o elimina datos de otras. Todo esto bajo un proceso controlado mediante una transacción, la cual puede deshacerse si algo falla.

*BENEFICIOS*

* El motor de DB controla las operaciones.
* Se ejecuta en un servidor independiente.
* Devuelve al cliente el resultado final, evitando sobrecargar su computadora con procesos.
* Evita programar una lógica compleja del lado del cliente.
* Minimiza los errores concentrando las operaciones.

*TIPOS DE STORED PROCEDURES*

* Procedimiento Almacenado básico.
* Procedimiento Almacenado con parámetro(s) de entrada.
* Procedimiento Almacenado con parámetro(s) de salida.
* Procedimiento Almacenado con parámetro(s) de entrada y salida.

*IMPLEMENTACIÓN EFICAZ DE UN S.P.*

Imaginemos una empresa que venda un producto con alta demanda de público, existen varios canales de venta y un stock de producto limitado.

¿Cómo implementar un Stored Procedure efectivo?

Cada canal de venta (*telefónico, físico, e-commerce*), invoca al Stored Procedure para registrar una venta del producto.

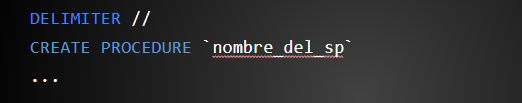
El S.P. busca el **precio** actualizado, crea la **factura** de venta, descuenta el **stock**, informa a **logística** para el despacho, registra el movimiento en la tabla de **LOG**, asigna la **comisión** al vendedor, registra en el área **contable** la ganancia por venta.*Todas estas transacciones en diferentes tablas deben ser consistentes. Si uno de los puntos de estos procesos fallase, debería deshacerse todo lo anterior.*

👉 Eso es lo que podemos lograr con un Stored Procedure, de forma fácil y controlada, evitando que un error de código de una aplicación web o móvil, rompa la consistencia de datos.

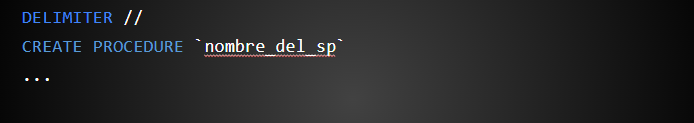
*SINTAXIS*

La sintaxis de un Stored Procedure se asimila en muchos puntos a lo que vimos la clase pasada (*Functions*). Pero, a diferencia de las funciones, un S.P. puede o no tener parámetros de entrada, salida, y/o combinar ambos.

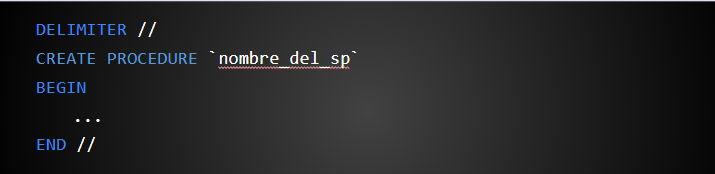
Se inicia con la sentencia **DELIMITER**, seguida de un conjunto de caracteres que no usarás dentro del S.P.



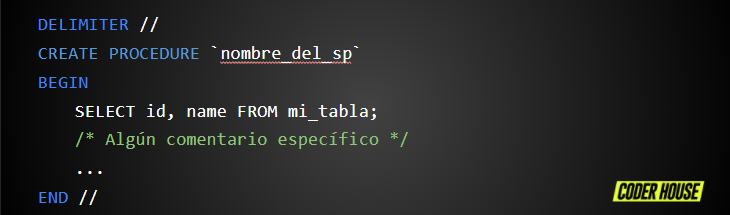
Luego define la sentencia **CREATE PROCEDURE**, seguida del nombre del mismo.



Integra **BEGIN** y **END**, para determinar dónde inicia y finaliza el código del procedimiento almacenado.



Y dentro de **BEGIN** y **END**, el código que le dará vida a tu procedimiento almacenado.

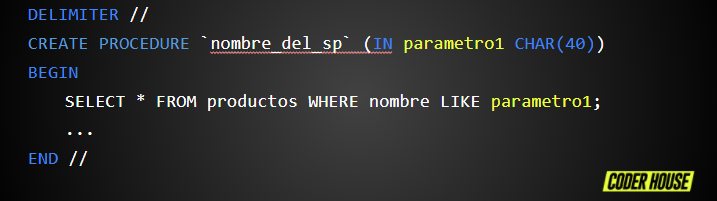


*PARÁMETROS DE ENTRADA*

Dentro de un S.P., podemos definir parámetros de entrada, los cuales recibirán valores cuando ejecutamos el S.P., de la misma forma que le enviamos parámetros a una Función SQL.

Para definirlos, debemos utilizar la palabra reservada **IN**, y especificar el tipo de dato que soportan.

Definido el o los parámetros de entrada en el encabezado, podrás utilizarlos luego dentro del código.

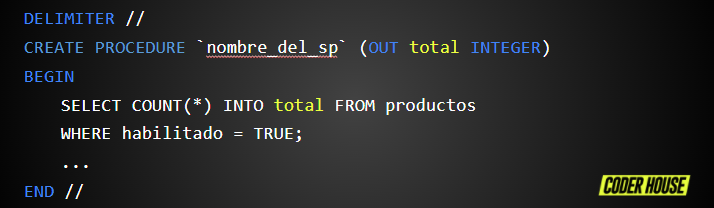


*PARÁMETROS DE SALIDA*

En el S.P. puedes definir también parámetros de salida, los cuales funcionan como un cursos, recibiendo valores directamente del código del S.P.

Para definirlos, debemos utilizar la palabra reservada **OUT**, y especificando también el tipo de dato.

Definido el o los parámetros de entrada en el encabezado, podrás utilizarlos luego dentro del código.

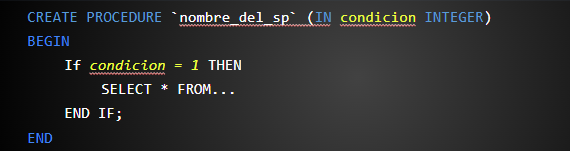


*INTEGRAR CONDICIONALES*

*CONTROLAR LA EJECUCIÓN DEL S.P. MEDIANTE EL USO DE CONDICIONALES*

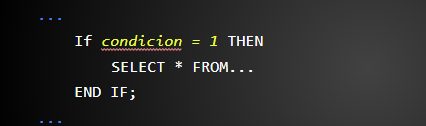
*INTEGRAR CONDICIONALES*

El lenguaje de programación **SQL** soporta el uso de condicionales, como cualquier otro lenguaje de programación.



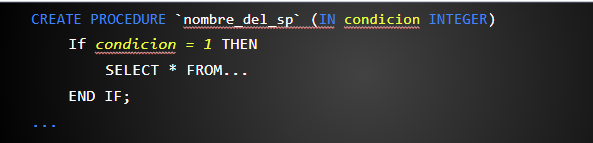
*IF ... THEN*

**IF** ... **THEN** es el condicional más utilizado y común a todos los lenguajes de programación. El código que se defina dentro de este bloque solo se ejecutará si se cumple la expresión que estamos evaluando (condicion = 1).



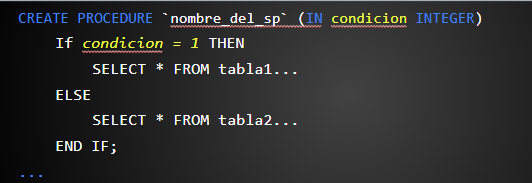
*INTEGRAR CONDICIONALES*

En este caso, condicion es una variable y el valor que tenga asignado puede provenir como un parámetro **IN** del Stored Procedure o como el resultado de un cálculo o consulta interna, realizados dentro del S.P.



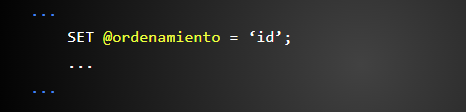
*ELSE*

**ELSE** se utiliza solo en combinación con **IF** y permite definir un path o ruta alternativa de ejecución, si la condición evaluada por IF, no se cumple.



*DEFINIR VARIABLES DENTRO DE UN S.P.*

Tal como vimos con las funciones, las variables también pueden ser declaradas dentro de un S.P. Esto lo realizamos utilizando la palabra reservada **SET**, seguida del nombre de la variable, que debe incluir el **@** en primer lugar.



*CONVERTIR STRINGS SQL EN CLÁUSULAS*

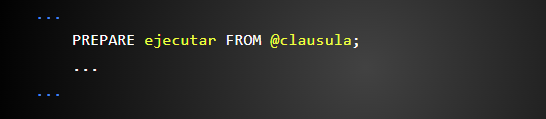
Si utilizamos cadenas de strings y/o variables para armar una estructura del tipo DML con el nombre de un campo, un valor a buscar, y/o una consulta SQL, deberemos concatenar todo previo a ejecutar dicha instrucción SQL.

Para que todo esto funcione, debemos integrar en el S.P. **PREPARE**, **EXECUTE** y **DEALLOCATE**.

**PREPARE** convierte una cláusula SQL a un elemento u objeto que el Motor SQL podrá interpretar mejor. Las cláusulas SQL escritas en una ventana de script o variable SQL no son más que una cadena de string.

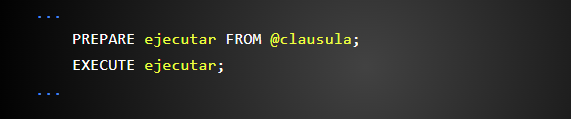
El motor SQL necesita un objeto SQL para entenderlo y ejecutarlo, y es allí donde PREPARE entra en acción.

La cláusula SQL debe almacenarse en una variable específica, declarada junto a la sentencia PREPARE, referenciando la misma con el origen de la cláusula SQL en formato string.



*EJECUTAR LA CLÁUSULA*

Finalmente, con la cláusula preparada como “un objeto entendible por el motor SQL”, solo nos queda ejecutarla utilizando la sentencia **EXECUTE**.



Una vez ejecutada la sentencia SQL, podemos deshacer el objeto interpretable por el motor SQL, invocando la sentencia **DEALLOCATE PREPARE**.

