



Clase 16. Curso SQL

# ***STORED PROCEDURES***

***RECUERDA PONER A GRABAR LA  
CLASE***





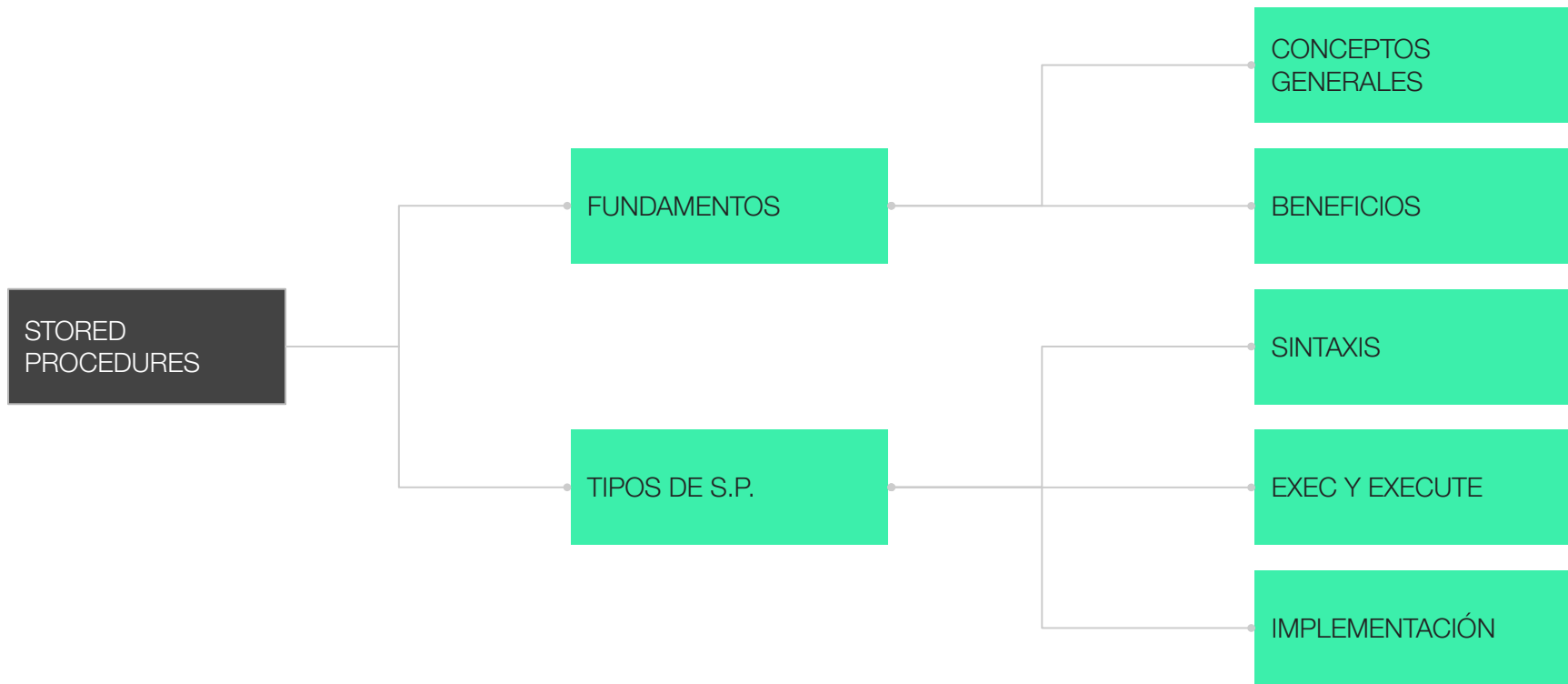
## ***OBJETIVOS DE LA CLASE***

- Reconocer un stored procedure.
- Identificar los tipos de stored procedure.
- Implementar store procedures.

# ***MAPA DE CONCEPTOS***

# MAPA DE CONCEPTOS CLASE 16

¡Para  
recordar!



# ***STORED PROCEDURES***

# ***CONCEPTO GENERAL***



# ***STORED PROCEDURE***

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.





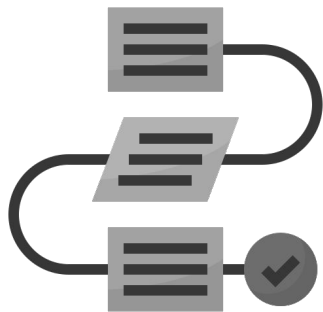
# ***STORED PROCEDURE***



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.



# ***STORED PROCEDURE***



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.



# ***STORED PROCEDURE***



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***

***CODER HOUSE***



# ***STORED PROCEDURE***



- 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***

# ***TIPOS DE STORED PROCEDURE***



Dentro de los tipos de **Stored Procedures**, encontramos:

- 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.***





# ***IMPLEMENTACIÓN EFICAZ***

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?

# ***IMPLEMENTACIÓN EFICAZ***



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.

# IMPLEMENTACIÓN EFICAZ



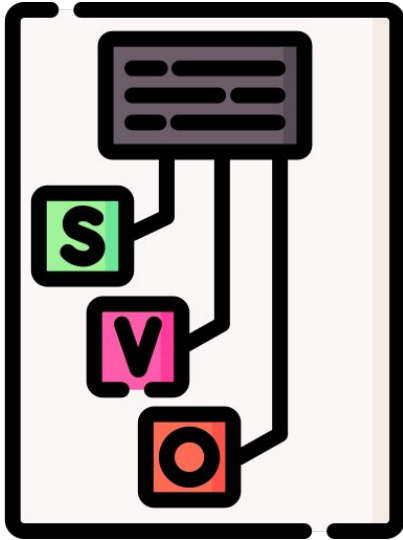
*Todas estas transacciones en diferentes tablas deben ser consistentes. Si uno de los puntos de estos procesos fallase, debería deshacerse todo lo anterior.*

👉 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***

***CODER HOUSE***

# ***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.



# ***SINTAXIS BÁSICA***

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

```
DELIMITER //
```

```
CREATE PROCEDURE `nombre_del_sp`
```

```
...
```



# ***SINTAXIS BÁSICA***

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

```
DELIMITER //  
CREATE PROCEDURE `nombre_del_sp`  
...
```



# ***SINTAXIS BÁSICA***

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

```
DELIMITER //  
CREATE PROCEDURE `nombre_del_sp`  
BEGIN  
    ...  
END //
```





# ***SINTAXIS BÁSICA***

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

```
DELIMITER //  
CREATE PROCEDURE `nombre_del_sp`  
BEGIN  
    SELECT id, name FROM mi_tabla;  
    /* Algún comentario específico */  
    ...  
END //
```

# ***PARÁMETROS DE ENTRADA***

# ***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.

# PARÁMETROS DE ENTRADA



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

```
DELIMITER //
```

```
CREATE PROCEDURE `nombre_del_sp` (IN parametro1 CHAR(40))
```

```
BEGIN
```

```
    SELECT * FROM productos WHERE nombre LIKE parametro1;
```

```
    ...
```

```
END //
```

# ***PARÁMETROS DE SALIDA***

# ***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.

# ***PARÁMETROS DE SALIDA***



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

```
DELIMITER //
```

```
CREATE PROCEDURE `nombre_del_sp` (OUT total INTEGER)
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO total FROM productos
```

```
    WHERE habilitado = TRUE;
```

```
    ...
```

```
END //
```



## ***EJEMPLO EN VIVO***

*Veamos cómo implementar Stored Procedure simple.*





# ***S.P. SIMPLE***

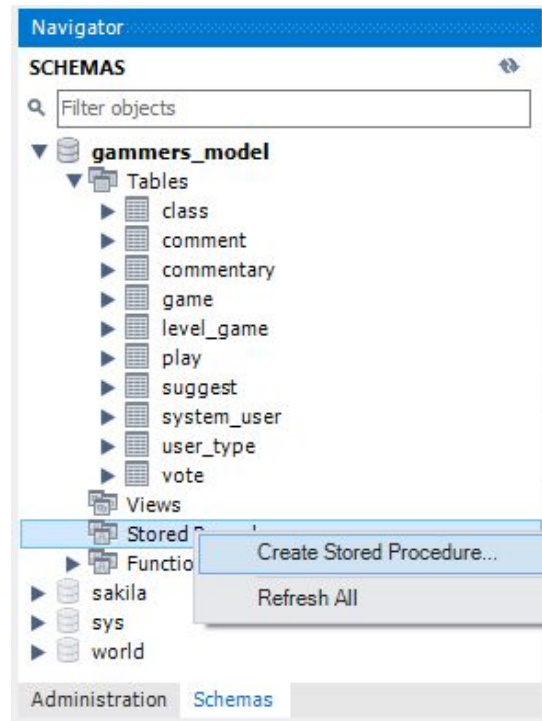
Desarrollaremos a continuación  
un **Procedimiento**  
**Almacenado** simple.  
El mismo listará los videojuegos  
que se encuentran en la tabla  
game.

	id_game	name	description	id_level	id_class
▶	1	Forza Horizon 5	odio donec	2	143
	2	Call of Duty: Vanguard	morbi non	6	153
	3	Shin Megami Tensei 5	turpis integer aliquet massa id	3	243
	4	Marvels Guardianes de la Galaxia	lobortis sapien sapien non mi	4	245
	5	Age of Empires IV		2	50
	6	Football Manager 22	nulla suspendisse potenti	8	236
	7	Football Manager 22	mauris lacinia sapien quis libero	11	173
	8	Blue Reflection: Second Light	libero rutrum ac lobortis	2	18
	9	Darkest Dungeon II	erat nulla	11	90
	10	Voice of Cards	parturient montes nascetur	2	275
	11	Elden Ring	et ultrices posuere cubilia curae	15	208
	12	FIFA 22: Ultimate Team	suspendisse	14	205



# ***S.P. SIMPLE***

En el apartado **Stored Procedures** del esquema gamers, hacemos clic con el botón secundario del mouse y seleccionamos del menú contextual, la opción **Create Stored Procedure...**

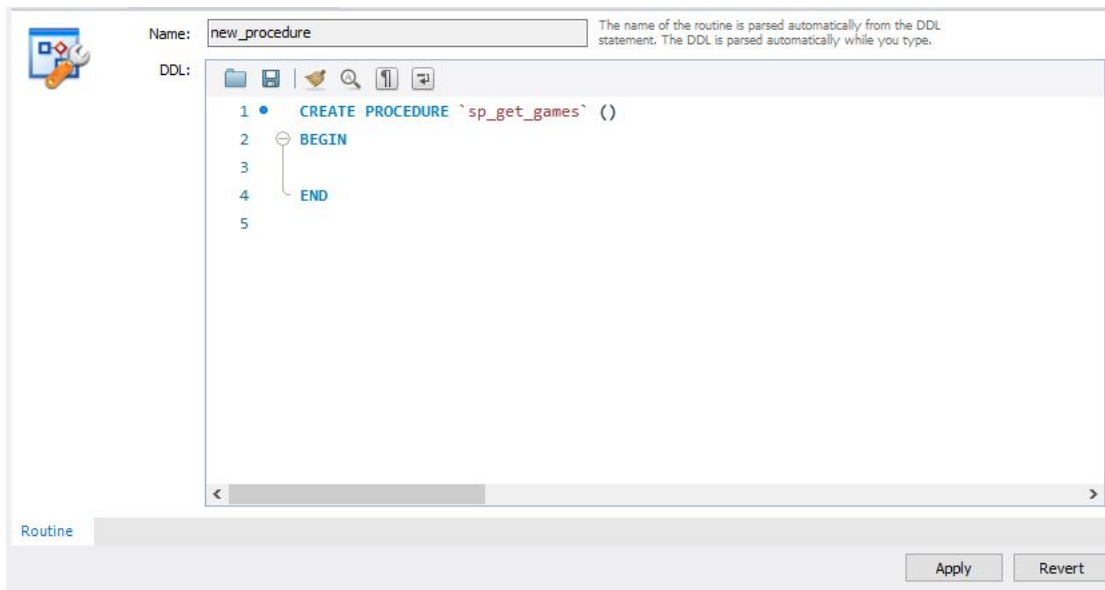




## S.P. SIMPLE

Se creará una pestaña nueva en **Mysql Workbench** con la estructura básica del **Stored Procedure**.

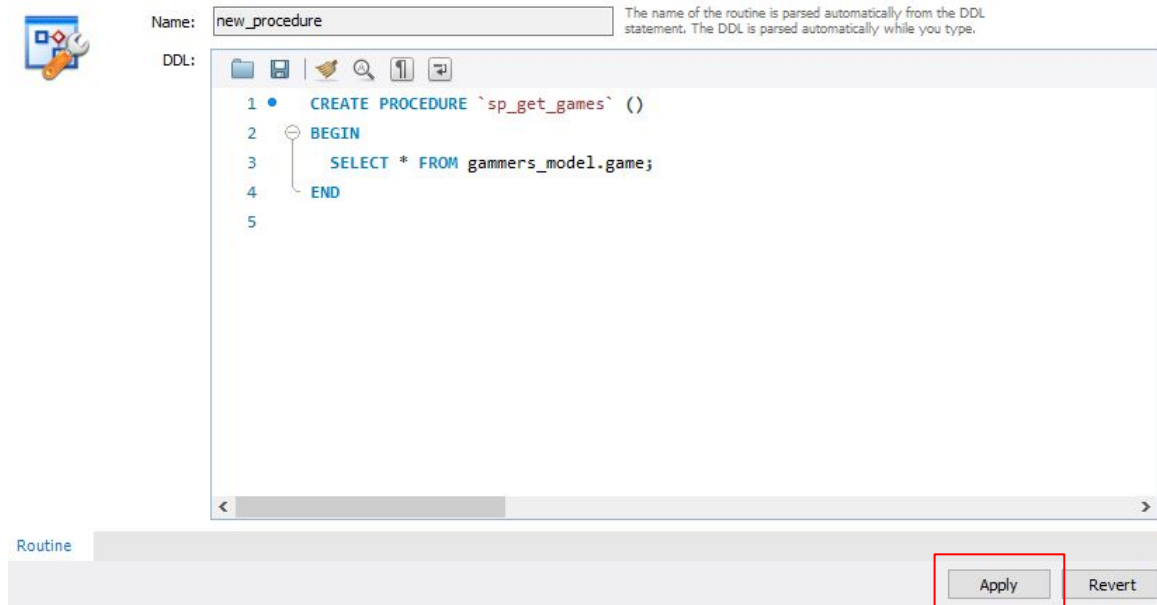
Reemplazamos el nombre por defecto, por el de **sp\_get\_games**.





## S.P. SIMPLE

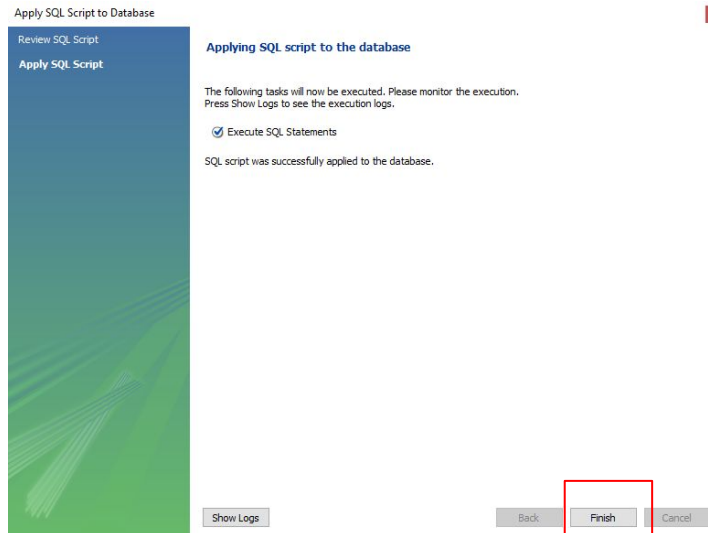
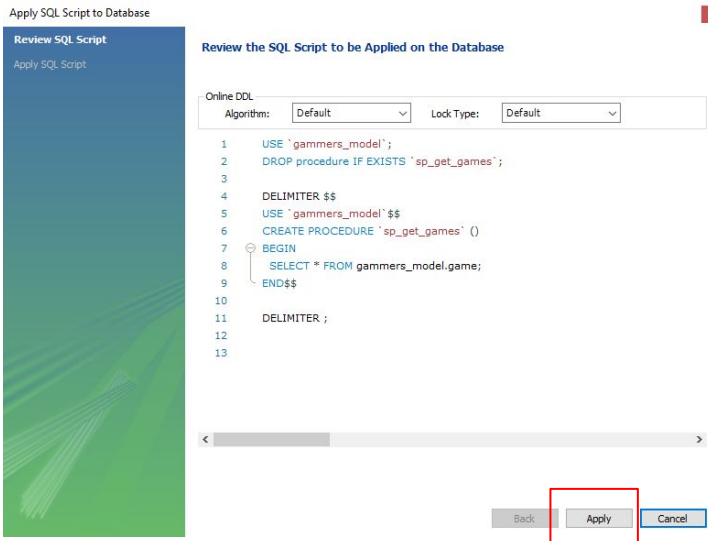
Agregamos dentro del apartado **BEGIN...END**, la sentencia SQL SELECT que deseamos ejecutar.





# S.P. SIMPLE

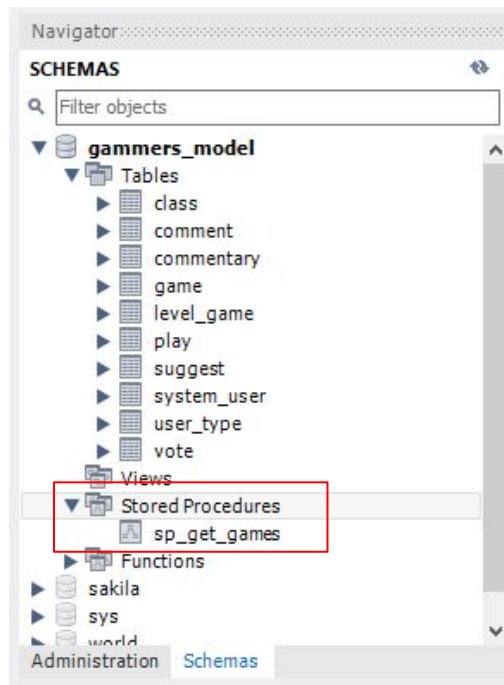
Pulsamos el botón **Apply**, ubicado en el extremo inferior derecho del panel donde estamos trabajando. En la ventana emergente, verificamos que todo el código sea correcto, y volvemos a pulsar el botón **Apply**.





# ***S.P. SIMPLE***

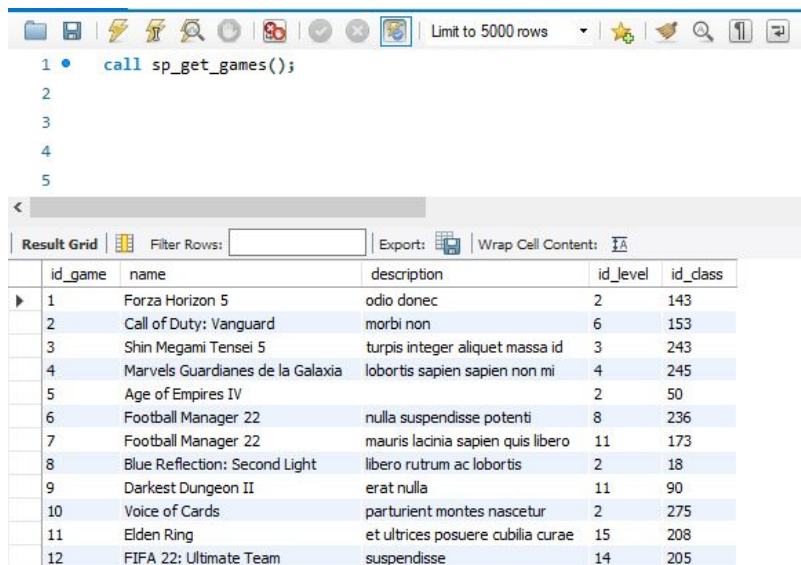
Dentro del sección de Stored procedures encontraremos nuestro S.P. creado:





# S.P. SIMPLE

Finalmente nos queda invocar al **Stored Procedure** desde una ventana de Script, escribiendo la sentencia **call**, seguida del nombre del S.P.



```
1 • call sp_get_games();
2
3
4
5
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	id_game	name	description	id_level	id_class
▶	1	Forza Horizon 5	odio donec	2	143
	2	Call of Duty: Vanguard	morbi non	6	153
	3	Shin Megami Tensei 5	turpis integer aliquet massa id	3	243
	4	Marvels Guardianes de la Galaxia	lobortis sapien sapien non mi	4	245
	5	Age of Empires IV		2	50
	6	Football Manager 22	nulla suspendisse potenti	8	236
	7	Football Manager 22	mauris lacinia sapien quis libero	11	173
	8	Blue Reflection: Second Light	libero rutrum ac lobortis	2	18
	9	Darkest Dungeon II	erat nulla	11	90
	10	Voice of Cards	parturient montes nascetur	2	275
	11	Elden Ring	et ultrices posuere cubilia curae	15	208
	12	FIFA 22: Ultimate Team	suspendisse	14	205



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**



# ***INTEGRAR CONDICIONALES***

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

# ***INTEGRAR CONDICIONALES***

Ejemplo  
en vivo



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

```
CREATE PROCEDURE `nombre_del_sp` (IN condicion INTEGER)
BEGIN
    If condicion = 1 THEN
        SELECT * FROM...
    END IF;
END
```

***CODER HOUSE***

# ***IF ... THEN***

Ejemplo  
en vivo



**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`).

...

```
If condicion = 1 THEN  
    SELECT * FROM...  
END IF;
```

...

***CODER HOUSE***

# INTEGRAR CONDICIONALES

Ejemplo  
en vivo



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.

```
CREATE PROCEDURE `nombre_del_sp` (IN condicion INTEGER)
  If condicion = 1 THEN
    SELECT * FROM...
  END IF;
...
```



# 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.

```
CREATE PROCEDURE `nombre_del_sp` (IN condicion INTEGER)
  If condicion = 1 THEN
    SELECT * FROM tabla1...
  ELSE
    SELECT * FROM tabla2...
  END IF;
```

...

***DEFINIR VARIABLES DENTRO DE UN S.P.***

# ***DECLARAR VARIABLES EN 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.

```
...  
SET @ordenamiento = 'id';  
...  
...
```



# ***CONVERTIR STRINGS SQL EN CLÁUSULAS***

# ***CONVERTIR STRINGS 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**.

# ***CONVERTIR STRINGS EN CLÁUSULAS***



**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.

# ***CONVERTIR STRINGS EN CLÁUSULAS***

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.

...

```
PREPARE ejecutar FROM @clausula;
```

...

...

***EJECUTAR LA CLÁUSULA***

# ***EJECUTAR LA CLÁUSULA***

Ejemplo  
en vivo



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

...

```
PREPARE ejecutar FROM @clausula;  
EXECUTE ejecutar;
```

...



# ***EJECUTAR LA CLÁUSULA***

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

...

```
PREPARE ejecutar FROM @clausula;  
EXECUTE ejecutar;  
DEALLOCATE PREPARE ejecutar;
```

...



## ***EJEMPLO EN VIVO***

*Veremos cómo implementar PREPARE y EXECUTE en un  
Stored procedure.*





# ***IMPLEMENTAR PREPARE Y EXECUTE***

Creamos un nuevo **Stored Procedure**, se llamará **sp\_get\_games\_order**. Recibirá un parámetro que hará referencia a un campo de la tabla, el cual indicará el orden de la consulta a mostrar.

```
CREATE PROCEDURE `sp_get_games_order` (IN field CHAR(20))  
  BEGIN  
    ...  
  END
```



# ***IMPLEMENTAR PREPARE Y EXECUTE***

Comparamos si el parámetro de entrada **field**, incluye o no un valor. Si lo incluye, definimos una variable **@game\_order** con la cláusula **ORDER BY**.

Si no, definimos la misma variable, pero vacía.

```
IF field <> '' THEN
    SET @game_order = concat('ORDER BY ', field));
ELSE
    SET @game_order = '';
END IF;
```



# ***IMPLEMENTAR PREPARE Y EXECUTE***

Declaremos una nueva variable **@clausula**, donde concatenamos la sentencia SQL junto al ordenamiento establecido en la variable **@game\_order**.

Luego invocamos la cláusula **PREPARE** para convertir la cadena SQL en un objeto entendible por el motor de DB.

```
...  
END IF;  
SET @clausula = concat('SELECT * FROM game', @game_order);  
PREPARE runSQL FROM @clausula;  
EXECUTE runSQL;  
DEALLOCATE PREPARE runSQL;
```

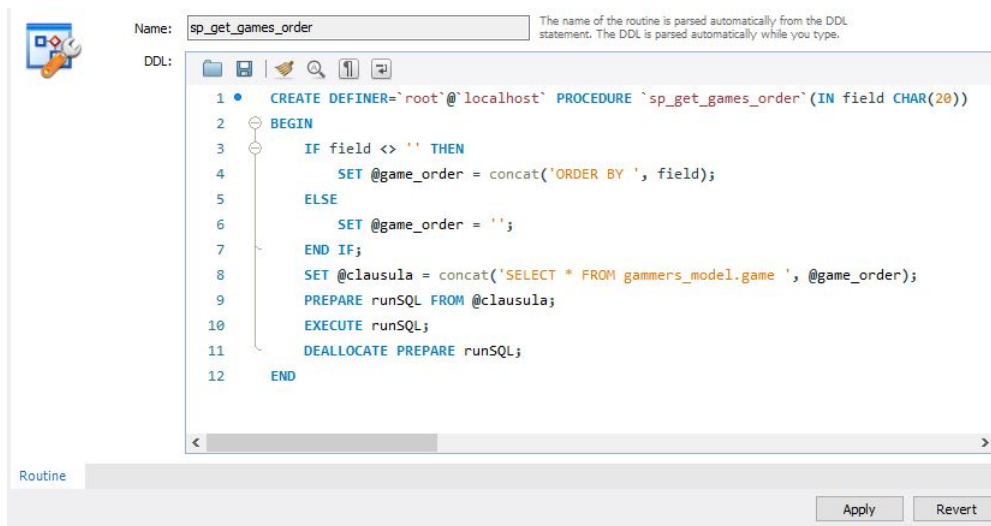
END



# IMPLEMENTAR PREPARE Y EXECUTE

Aplicamos los cambios del S.P. para que quede compilado y guardado en el motor de BD.

Luego, en una ventana de script, invocamos el Stored Procedure, pasándole como parámetro uno de los campos de la tabla a consultar.



The screenshot shows a database script editor window. At the top, there's a 'Name:' field containing 'sp\_get\_games\_order' and a tooltip that says 'The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.' Below this is a 'DDL:' section with a toolbar and a list of SQL statements. The statements are as follows:

```
1 CREATE DEFINER='root'@'localhost' PROCEDURE `sp_get_games_order` (IN field CHAR(20))
2 BEGIN
3     IF field <> '' THEN
4         SET @game_order = concat('ORDER BY ', field);
5     ELSE
6         SET @game_order = '';
7     END IF;
8     SET @clausula = concat('SELECT * FROM gammers_model.game ', @game_order);
9     PREPARE runSQL FROM @clausula;
10    EXECUTE runSQL;
11    DEALLOCATE PREPARE runSQL;
12 END
```

At the bottom of the window, there's a 'Routine' tab and two buttons: 'Apply' and 'Revert'.



# IMPLEMENTAR PREPARE Y EXECUTE



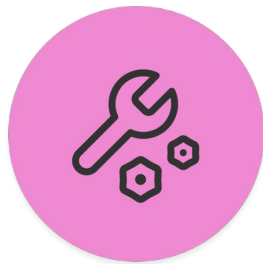
```
1 • call sp_get_games_order("name");  
2  
3  
4  
5
```

<					
Result Grid   Filter Rows:   Export:   Wrap Cell Content:					
	id_game	name	description	id_level	id_class
▶	97	Actraiser Renaissance	viverra eget congue	3	231
	5	Age of Empires IV		2	50
	46	Alan Wake Remastered	dui luctus rutrum nulla tellus	15	137
	30	Assassins Creed Valhalla	vivamus	9	207
	67	Aven Colony	sollicitudin ut	14	238
	74	Back 4 Blood	pulvinar	9	36
	48	Back 4 Blood	ornare imperdiet sapien urna pretium	4	198
	59	Battlefield 2042	pede libero	12	31
	51	Battlefield 2042		6	153
	93	Blast Brigade	in sapien iaculis congue vivamus	8	19
	8	Blue Reflection: Second...	libero rutrum ac lobortis	2	18
	91	Bright Memory: Infinite		5	77

# ***FORMAS DE EJECUTAR UN S.P.***

Al principio podrás confundirte usando los comandos EXECUTE, CALL y EXEC. El primero se usa exclusivamente dentro del código de un Stored Procedure.

**CALL** se usa en una ventana de Script para invocar un Stored Procedure y, **EXEC**, en Mysql no se utiliza, dado que es propio de SQL Server para “llamar” un S.P.



# ***S.P. de INSERCIÓN DE REGISTROS***

Implementaremos un Stored Procedure de inserción de registros.

*Tiempo estimado: 15 minutos*

# ***S.P. de INSERCIÓN DE REGISTROS***

Desafío  
generico



Crea un Stored Procedure que inserte datos en una tabla.

1. Debe recibir un parámetro del tipo **char(xx)**
2. Inserta dicho parámetro como un nuevo registro en la tabla
3. Ejecuta luego, un **SELECT** sobre la tabla ordenada de forma descendente, para ver el registro insertado en primer lugar
4. Si el parámetro **char()** recibido es igual a '', devuelve un error que diga  
**'ERROR: no se pudo crear el producto indicado'**





# ***SCRIPT DE CREACIÓN DE STORED PROCEDURES***

Presentar en formato .sql el script de creación de dos Stored Procedures con base en los datos de la base de datos del proyecto final.

# SCRIPT DE CREACIÓN DE STORED PROCEDURES

**Formato:** El archivo a presentar debe ser del tipo .sql nombrado como "Stored+Apellido".

Desafío  
entregable



**>> Consigna:** Sobre las tablas creadas anteriormente para tu proyecto final, **agregar 2 Stored Procedures** que te permitan trabajar sobre las mismas.

**>>Aspectos a incluir en el entregable:**

El primer S.P. debe permitir indicar a través de un parámetro el campo de ordenamiento de una tabla y mediante un segundo parámetro, si el orden es descendente o ascendente.

El otro S.P. que crearás, puede (1: insertar registros en una tabla de tu proyecto. 2: eliminar algún registro específico de una tabla de tu proyecto.)

- Agrega comentarios en ambos Scripts de los S.P. para saber qué hacen y cómo usarlos
- Procura guiarte con los ejemplos abordados en esta clase

***¿PREGUNTAS?***



# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Fundamentos de Stored Procedures
- Cómo crear un Stored Procedure
- Parámetros de entrada y salida
- Variables, cláusulas, condicionales en S.P.



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***