BASES DE DATOS II

Stored Procedures

Repaso completo: sintaxis, parámetros, lógica condicional y bucles

UTN Mar del Plata
TUP — 2do Año | Plan 2024

Este material cubre el repaso de SPs del año anterior.
El manejo de errores y transacciones se trata en un documento aparte.

01 Repaso — ¿Qué es un Procedimiento Almacenado?

Un procedimiento almacenado (SP) es un bloque de instrucciones SQL que guardamos en el servidor de base de datos con un nombre. Una vez creado, lo ejecutamos con CALL nombre(); desde cualquier cliente o aplicación, sin tener que reescribir el código.

El año pasado creamos SPs simples que hacían consultas e inserciones. Este año les agregamos la capacidad de manejar errores y garantizar que las operaciones sean seguras incluso cuando algo falla.

Ventaja	¿Qué significa en la práctica?
Rendimiento	Se precompilan en el servidor. Menos tráfico de red.
Seguridad	El usuario ejecuta el SP sin ver ni tocar las tablas directamente.
Reutilización	Una sola definición, miles de llamadas desde donde quieras.
Mantenimiento	Corregís la lógica en un lugar y el cambio aplica a toda la app.

02 Repaso — Sintaxis, DELIMITER y Parámetros

Todo SP en MySQL sigue la misma estructura. Lo más importante a recordar: necesitamos cambiar el DELIMITER temporalmente porque el cuerpo del SP usa ; internamente, y sin el cambio el cliente confunde esos puntos y coma con el fin del comando.

estructura_base.sql

```
DELIMITER //              -- cambiamos el delimitador
CREATE PROCEDURE nombre(parametros)
BEGIN
    -- instrucciones SQL
    -- pueden usar ; con tranquilidad acá adentro
END //                    -- fin del SP, usando el nuevo delimitador
DELIMITER ;               -- restauramos el delimitador original

CALL nombre(argumentos);  -- así lo ejecutamos
```

Parámetros IN, OUT e INOUT

Tipo	Uso	Para qué sirve	¿Cómo se llama / lee?
IN	Entrada	Pasar datos al SP desde afuera	CALL sp(valor)
OUT	Salida	Recibir un resultado producido por el SP	CALL sp(@v); SELECT @v
INOUT	Entrada + Salida	Pasar un valor y recibir uno modificado de vuelta	CALL sp(@v); SELECT @v

Ejemplo completo con IN y OUT

parametros_in_out.sql

```
DELIMITER //
CREATE PROCEDURE obtenerSaldo(
    IN  p_id_socio  INT,
    OUT p_saldo     DECIMAL(10,2)
)
BEGIN
    -- SELECT...INTO: asigna el resultado a la variable OUT
    SELECT saldo INTO p_saldo
    FROM cuentas
    WHERE id_socio = p_id_socio;
END //
DELIMITER ;

CALL obtenerSaldo(3, @s);   -- el resultado queda en @s
SELECT @s AS SaldoObtenido; -- lo leemos con SELECT
```

→ SELECT...INTO

Asigna el resultado de una consulta a una variable. Si la consulta no devuelve filas, la variable queda en NULL. Si devuelve más de una fila, MySQL lanza un error. Para devolver muchas filas, usás un SELECT normal (sin INTO) dentro del SP.

03 Repaso + Extensión — IF y CASE

Ya usamos IF para tomar decisiones dentro de un SP. Acá repasamos la sintaxis y agregamos CASE, que es la alternativa más legible cuando tenemos muchas condiciones posibles.

IF / ELSEIF / ELSE

if_repaso.sql

```
CREATE PROCEDURE categorizarSocio(IN p_antiguedad INT, OUT p_categoria VARCHAR(20))
BEGIN
    IF p_antiguedad >= 5 THEN
        SET p_categoria = 'Premium';
    ELSEIF p_antiguedad >= 2 THEN
        SET p_categoria = 'Regular';
    ELSE
        SET p_categoria = 'Nuevo';
    END IF;
END //
```

CASE — dos sintaxis posibles

CASE es el IF de muchas opciones. Cuando tenemos 4 o más ramas posibles, CASE hace el código mucho más fácil de leer y mantener.

case_dos_formas.sql

```
-- FORMA 1: CASE BUSCADO (Searched) — cada WHEN es una condición independiente
-- Es la más versátil: acepta rangos, AND, OR, etc.
CASE
    WHEN p_antiguedad >= 5 THEN SET p_categoria = 'Premium';
    WHEN p_antiguedad >= 2 THEN SET p_categoria = 'Regular';
    ELSE                        SET p_categoria = 'Nuevo';
END CASE;

-- FORMA 2: CASE SIMPLE — compara una variable contra valores exactos
-- Más conciso, pero NO sirve para rangos
CASE p_estado_plan
    WHEN 1 THEN SET p_descripcion = 'Activo';
    WHEN 2 THEN SET p_descripcion = 'Suspendido';
    WHEN 3 THEN SET p_descripcion = 'Vencido';
    ELSE        SET p_descripcion = 'Desconocido';
END CASE;
```

	CASE Buscado (Searched)	CASE Simple
Condición en WHEN	Expresión booleana (>=, AND, OR...)	Valor exacto (1, 2, texto...)
¿Acepta rangos?	Sí	No
¿Cuándo usarlo?	Cuando las ramas tienen lógica distinta o rangos	Cuando se compara una variable contra valores fijos

✓ **¿Cuándo conviene CASE en lugar de IF?**

Conviene CASE cuando todas las ramas comparan la misma variable o condición base — el código queda mucho más limpio y fácil de seguir.

04 Nuevo — Bucles: WHILE, REPEAT y LOOP

Los bucles permiten repetir un bloque de código mientras se cumpla una condición. MySQL ofrece tres variantes. La diferencia clave está en cuándo se evalúa la condición de corte.

WHILE — verificar antes de entrar

La condición se evalúa ANTES de ejecutar el cuerpo. Si la condición es falsa desde el inicio, el cuerpo no se ejecuta ni una vez.

while_ejemplo.sql

```
CREATE PROCEDURE contarHasta(IN p_limite INT)
BEGIN
    DECLARE v_i INT DEFAULT 1;  -- variable local, empieza en 1

    WHILE v_i <= p_limite DO
        SELECT v_i AS Numero;   -- muestra el valor actual
        SET v_i = v_i + 1;      -- incrementa (sin esto: bucle infinito)
    END WHILE;
END //

CALL contarHasta(5);  -- muestra 1, 2, 3, 4, 5
```

REPEAT — verificar después de ejecutar

El cuerpo se ejecuta SIEMPRE al menos una vez. La condición de corte se evalúa AL FINAL de cada iteración.

repeat_ejemplo.sql

```
CREATE PROCEDURE contarRepeat(IN p_limite INT)
BEGIN
    DECLARE v_i INT DEFAULT 1;

    REPEAT
        SELECT v_i AS Numero;
        SET v_i = v_i + 1;
    UNTIL v_i > p_limite        -- cuando esto sea verdadero, para
    END REPEAT;
END //

-- Diferencia clave: si p_limite = 0,
-- WHILE no ejecuta nada, REPEAT ejecuta el cuerpo una vez igual.
```

LOOP — salida explícita con LEAVE

LOOP no tiene condición propia: es un bucle infinito por definición. La salida se controla con LEAVE + una etiqueta. Útil cuando la condición de corte es compleja o puede darse en varios puntos.

loop_ejemplo.sql

```
CREATE PROCEDURE contarLoop(IN p_limite INT)
BEGIN
    DECLARE v_i INT DEFAULT 1;

    miLoop: LOOP                -- 'miLoop' es la etiqueta del bucle
        SELECT v_i AS Numero;
        SET v_i = v_i + 1;
        IF v_i > p_limite THEN
            LEAVE miLoop;       -- sale del bucle (equivale a 'break')
        END IF;
    END LOOP miLoop;
END //
```

Bucle	Evalúa condición	Mínimo ejecuciones	¿Cuándo usarlo?
WHILE	Antes de entrar al cuerpo	0 — puede no ejecutarse nunca	Cuando la condición puede ser falsa desde el inicio
REPEAT	Al final de cada iteración	1 — siempre ejecuta al menos una vez	Cuando el cuerpo siempre debe ejecutarse al menos una vez
LOOP	No tiene — se usa LEAVE	1 (infinito sin LEAVE)	Cuando la condición de salida es compleja o hay múltiples puntos de salida

! El error más común con bucles

Olvidar incrementar la variable contadora (SET v_i = v_i + 1). Sin esa línea el bucle nunca termina y MySQL quedará ejecutando indefinidamente hasta que lo interrumpas manualmente.

05 Repaso rápido — Variables en un SP

Conocemos dos tipos de variables que se usan dentro de un SP. Las repasamos brevemente porque las vamos a necesitar seguido en los temas que siguen, especialmente cuando trabajemos con handlers y transacciones.

Tipo	Declaración	Alcance	¿Para qué usarla?
Local (DECLARE)	DECLARE x TIPO; DECLARE x TIPO DEFAULT val;	Solo dentro del SP (BEGIN...END)	Cálculos intermedios, resultados temporales dentro del procedimiento.
Sesión (@var)	SET @x = valor; (sin DECLARE previo)	Toda la sesión de conexión	Leer el OUT del SP desde afuera. Compartir datos entre llamadas.

Las tres formas de asignar valor a una variable

variables_asignacion.sql

```
-- 1. SET: asignación directa, para valores simples o cálculos
SET @descuento = 0.15;
SET v_total = v_precio * (1 - @descuento);

-- 2. SELECT...INTO: trae un valor desde la base de datos
DECLARE v_saldo DECIMAL(10,2);
SELECT saldo INTO v_saldo FROM cuentas WHERE id_socio = 7;

-- 3. Parámetro OUT del SP: el 'return' de MySQL
-- El llamador lo lee con SELECT @variable después del CALL
SET p_resultado = v_saldo * @descuento;
```

✓ Regla rápida

Usás SET cuando sabés el valor de antemano o lo calculás vos. Usás SELECT...INTO cuando el valor tiene que venir de una consulta a la BD. Usás un parámetro OUT cuando querés que el llamador pueda leer el resultado afuera del SP.

→ ¿Qué sigue?

Este documento cubre el repaso de los fundamentos de SPs: sintaxis, parámetros, lógica condicional, bucles y variables. Son los cimientos sobre los que se construyen los temas nuevos de este año.

El paso siguiente es el manejo de errores con handlers, SIGNAL y transacciones, que se desarrollan en el documento complementario de la materia.

✓ Para el estudio en casa

La mejor forma de fijar estos conceptos es escribir los ejemplos en Workbench y ejecutarlos. Modificá los valores de los parámetros, rompé cosas a propósito y observá qué pasa. No hay mejor explicación que ver el resultado en pantalla.