SERVICIO NACIONAL DE APRENDIZAJE SENA

Consultas base de datos Sakila

Título: Parte 2 para de consultas a la base de datos Sakila

Jaider Sebastián Moreno Quintero

03/07/2025

Tabla de contenido

Desarrollo de las consultas.....¡Error! Marcador no definido.

Introducción

El presente informe tiene como fin dar desarrollo a la guía de 10 consultas SQL para la base de datos SAKILA en mysql, las consultas a desarrollar son:

1. Crear procedimiento para alquilar una película
2. Procedimiento para generar reporte de ventas
3. Trigger para actualizar last_update automáticamente
4. Trigger para registrar cambios en películas
5. Función para calcular multas por retraso
6. Función para verificar disponibilidad de película
7. Calcular días de alquiler restantes
8. Procedimiento para devolver una película
9. Trigger para actualizar disponibilidad después de alquilar
10. Trigger para Validar pago antes de devolución

Adicionalmente para el proyecto formativo se tienen estas consultas a desarrollar:

11. 5 ejemplos de Stored Procedures aplicados al proyecto formativo.
12. 5 ejemplos de Functions aplicados al proyecto formativo.
13. 5 ejemplo de Triggers aplicados al proyecto formativo.
14. Formas de encriptación de contraseña en SQL.

1. Crear procedimiento para alquilar una película

```
DELIMITER $$

CREATE PROCEDURE alquilar_pelicula(
    IN p_customer_id INT,
    IN p_inventory_id INT,
    IN p_staff_id INT
)
BEGIN
    DECLARE rental_id INT;

    -- Insertar en tabla rental
    INSERT INTO rental (rental_date, inventory_id, customer_id, staff_id)
    VALUES (NOW(), p_inventory_id, p_customer_id, p_staff_id);

    SET rental_id = LAST_INSERT_ID();

    -- Insertar el pago
    INSERT INTO payment (customer_id, staff_id, rental_id, amount, payment_date)
    VALUES (p_customer_id, p_staff_id, rental_id, 2.99, NOW()); -- Asumiendo monto fijo

END$$

DELIMITER ;
```

2. Procedimiento para generar reporte de ventas

```
DELIMITER $$

CREATE PROCEDURE reporte_ventas()
BEGIN
    SELECT s.store_id, c.city, SUM(p.amount) AS total_ventas
    FROM payment p
    JOIN staff st ON p.staff_id = st.staff_id
    JOIN store s ON st.store_id = s.store_id
    JOIN address a ON s.address_id = a.address_id
    JOIN city c ON a.city_id = c.city_id
    GROUP BY s.store_id, c.city;
END$$

DELIMITER ;
```

3. Trigger para actualizar last_update automáticamente

```
DELIMITER $$

CREATE TRIGGER actualizar_last_update
BEFORE UPDATE ON film
FOR EACH ROW
BEGIN
    SET NEW.last_update = NOW();
END$$

DELIMITER ;
```

4. Trigger para registrar cambios en películas

```
DELIMITER $$

CREATE TABLE cambios_pelicula (
    cambio_id INT AUTO_INCREMENT PRIMARY KEY,
    film_id INT,
    titulo_viejo VARCHAR(255),
    titulo_nuevo VARCHAR(255),
    fecha_cambio DATETIME
);

CREATE TRIGGER registrar_cambio_pelicula
BEFORE UPDATE ON film
FOR EACH ROW
BEGIN
    IF OLD.title <> NEW.title THEN
        INSERT INTO cambios_pelicula (film_id, titulo_viejo, titulo_nuevo, fecha_cambio)
        VALUES (OLD.film_id, OLD.title, NEW.title, NOW());
    END IF;
END$$

DELIMITER ;
```

5. Función para calcular multas por retraso

```
DELIMITER $$

CREATE FUNCTION calcular_multa(return_date DATETIME, due_date DATETIME)
RETURNS DECIMAL(5,2)
DETERMINISTIC
BEGIN
    DECLARE dias_retraso INT;
    SET dias_retraso = DATEDIFF(return_date, due_date);
    RETURN IF(dias_retraso > 0, dias_retraso * 1.50, 0.00); -- Multa de $1.50 por día
END$$

DELIMITER ;
DELIMITER $$
```

6. Función para verificar disponibilidad de película

```
DELIMITER $$

CREATE FUNCTION pelicula_disponible(film_id INT)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE disponibles INT;
    SELECT COUNT(*) INTO disponibles
    FROM inventory i
    WHERE i.film_id = film_id
      AND i.inventory_id NOT IN (
          SELECT inventory_id FROM rental
          WHERE return_date IS NULL
      );
    RETURN disponibles > 0;
END$$

DELIMITER ;
```

7. Calcular días de alquiler restantes

```
DELIMITER $$

CREATE FUNCTION dias_alquiler_restantes(rental_id INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE rental_date DATETIME;
    DECLARE duration INT;

    SELECT r.rental_date, f.rental_duration
    INTO rental_date, duration
    FROM rental r
    JOIN inventory i ON r.inventory_id = i.inventory_id
    JOIN film f ON i.film_id = f.film_id
    WHERE r.rental_id = rental_id;

    RETURN duration - DATEDIFF(NOW(), rental_date);
END$$

DELIMITER ;
```

8. Procedimiento para devolver una película

```
DELIMITER $$

CREATE PROCEDURE devolver_pelicula(
    IN p_rental_id INT
)
BEGIN
    UPDATE rental
    SET return_date = NOW()
    WHERE rental_id = p_rental_id;
END$$

DELIMITER ;
```

9. Trigger para actualizar disponibilidad después de alquilar

```
DELIMITER $$

CREATE TRIGGER actualizar_disponibilidad_despues_alquilar
AFTER INSERT ON rental
FOR EACH ROW
BEGIN
    UPDATE inventory
    SET disponible = FALSE
    WHERE inventory_id = NEW.inventory_id;
END$$

DELIMITER ;
```

10. Trigger para Validar pago antes de devolución

```
DELIMITER $$

CREATE TRIGGER validar_pago_antes_devolucion
BEFORE UPDATE ON rental
FOR EACH ROW
BEGIN
    DECLARE pago_existente INT;

    IF NEW.return_date IS NOT NULL AND OLD.return_date IS NULL THEN
        SELECT COUNT(*) INTO pago_existente
        FROM payment
        WHERE rental_id = OLD.rental_id;

        IF pago_existente = 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'No se puede devolver la película sin realizar el pago.';
        END IF;
    END IF;
END$$

DELIMITER ;
```

11. Ejemplos de Stored Procedures aplicados al proyecto formativo.

```
CREATE PROCEDURE sp_registrar_pedido(
    IN p_mesa_id INT,
    IN p_cliente_id INT,
    IN p_usuario_id INT,
    IN p_notas TEXT
)
BEGIN
    DECLARE v_pedido_id INT;

    INSERT INTO pedidos (mesa_id, cliente_id, usuario_id, notas, estado)
    VALUES (p_mesa_id, p_cliente_id, p_usuario_id, p_notas, 'recibido');

    SET v_pedido_id = LAST_INSERT_ID();

    -- Actualizar estado de la mesa
    UPDATE mesas SET estado = 'ocupada' WHERE mesa_id = p_mesa_id;

    SELECT v_pedido_id AS nuevo_pedido_id;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE sp_agregar_platillo_pedido(
    IN p_pedido_id INT,
    IN p_platillo_id INT,
    IN p_cantidad INT,
    IN p_personalizaciones TEXT
)
BEGIN
    DECLARE v_precio DECIMAL(10,2);

    -- Obtener precio actual del platillo
    SELECT precio INTO v_precio FROM platillos WHERE platillo_id = p_platillo_id;

    -- Insertar detalle
    INSERT INTO detalles_pedido (pedido_id, platillo_id, cantidad, precio_unitario, personalizaciones)
    VALUES (p_pedido_id, p_platillo_id, p_cantidad, v_precio, p_personalizaciones);

    -- Actualizar total del pedido
    UPDATE pedidos
    SET total = (SELECT SUM(cantidad * precio_unitario) FROM detalles_pedido WHERE pedido_id = p_pedido_id)
    WHERE pedido_id = p_pedido_id;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE sp_ajustar_inventario(
    IN p_ingrediente_id INT,
    IN p_usuario_id INT,
    IN p_cantidad DECIMAL(10,2),
    IN p_tipo_movimiento ENUM('entrada', 'salida', 'ajuste'),
    IN p_motivo TEXT
)
BEGIN
    DECLARE v_costo DECIMAL(10,2);

    -- Obtener costo por unidad
    SELECT costo_por_unidad INTO v_costo FROM ingredientes WHERE ingrediente_id = p_ingrediente_id;

    -- Registrar movimiento
    INSERT INTO inventario (ingrediente_id, usuario_id, cantidad, tipo_movimiento, motivo, costo_total)
    VALUES (p_ingrediente_id, p_usuario_id, p_cantidad, p_tipo_movimiento, p_motivo, p_cantidad * v_costo);

    -- Actualizar stock
    IF p_tipo_movimiento = 'entrada' THEN
        UPDATE ingredientes
        SET stock_actual = stock_actual + p_cantidad
        WHERE ingrediente_id = p_ingrediente_id;
    ELSE
        UPDATE ingredientes
        SET stock_actual = stock_actual - p_cantidad
        WHERE ingrediente_id = p_ingrediente_id;
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE sp_reporte_ventas(
    IN p_fecha_inicio DATE,
    IN p_fecha_fin DATE
)
BEGIN
    SELECT
        p.platillo_id,
        pl.nombre AS platillo_nombre,
        SUM(p.cantidad) AS total_vendido,
        SUM(p.cantidad * p.precio_unitario) AS total_ingresos,
        c.nombre AS categoria
    FROM
        detalles_pedido p
        JOIN platillos pl ON p.platillo_id = pl.platillo_id
        JOIN categorias c ON pl.categoria_id = c.categoria_id
        JOIN pedidos pe ON p.pedido_id = pe.pedido_id
    WHERE
        DATE(pe.fecha_hora) BETWEEN p_fecha_inicio AND p_fecha_fin
    GROUP BY
        p.platillo_id, pl.nombre, c.nombre
    ORDER BY
        total_vendido DESC;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE sp_cambiar_estado_pedido(
    IN p_pedido_id INT,
    IN p_nuevo_estado ENUM('recibido', 'en preparacion', 'listo', 'entregado', 'cancelado', 'pagado')
)
BEGIN
    DECLARE v_mesa_id INT;

    -- Actualizar estado del pedido
    UPDATE pedidos SET estado = p_nuevo_estado WHERE pedido_id = p_pedido_id;

    -- Si el pedido se marca como pagado o cancelado, liberar la mesa
    IF p_nuevo_estado IN ('pagado', 'cancelado') THEN
        SELECT mesa_id INTO v_mesa_id FROM pedidos WHERE pedido_id = p_pedido_id;

        IF v_mesa_id IS NOT NULL THEN
            UPDATE mesas SET estado = 'disponible' WHERE mesa_id = v_mesa_id;
        END IF;
    END IF;
END //
DELIMITER ;
```

12. Ejemplos de Functions aplicados al proyecto formativo.

```
DELIMITER //
CREATE FUNCTION fn_verificar_disponibilidad_ingrediente(
    p_platillo_id INT,
    p_cantidad INT
) RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE v_disponible BOOLEAN DEFAULT TRUE;
    DECLARE v_ingrediente_id INT;
    DECLARE v_cantidad_necesaria DECIMAL(10,2);
    DECLARE v_stock_actual DECIMAL(10,2);
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT ingrediente_id, cantidad
        FROM platillo_ingredientes
        WHERE platillo_id = p_platillo_id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO v_ingrediente_id, v_cantidad_necesaria;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SELECT stock_actual INTO v_stock_actual
        FROM ingredientes
        WHERE ingrediente_id = v_ingrediente_id;

        IF (v_cantidad_necesaria * p_cantidad) > v_stock_actual THEN
            SET v_disponible = FALSE;
            LEAVE read_loop;
        END IF;
    END LOOP;
    CLOSE cur;

    RETURN v_disponible;
END //
DELIMITER ;
```

```sql
DELIMITER //
CREATE FUNCTION fn_tiempo_preparacion_pedido(
    p_pedido_id INT
) RETURNS INT
READS SQL DATA
BEGIN
    DECLARE v_tiempo_total INT DEFAULT 0;

    SELECT SUM(pl.tiempo_preparacion * dp.cantidad) INTO v_tiempo_total
    FROM detalles_pedido dp
    JOIN platillos pl ON dp.platillo_id = pl.platillo_id
    WHERE dp.pedido_id = p_pedido_id;

    RETURN IFNULL(v_tiempo_total, 0);
END //
DELIMITER ;
```

```sql
DELIMITER //
CREATE FUNCTION fn_platillo_mas_popular(
    p_dias INT
) RETURNS VARCHAR(100)
READS SQL DATA
BEGIN
    DECLARE v_nombre_platillo VARCHAR(100);

    SELECT pl.nombre INTO v_nombre_platillo
    FROM detalles_pedido dp
    JOIN platillos pl ON dp.platillo_id = pl.platillo_id
    JOIN pedidos p ON dp.pedido_id = p.pedido_id
    WHERE p.fecha_hora >= DATE_SUB(CURRENT_DATE(), INTERVAL p_dias DAY)
    GROUP BY dp.platillo_id, pl.nombre
    ORDER BY SUM(dp.cantidad) DESC
    LIMIT 1;

    RETURN IFNULL(v_nombre_platillo, 'No hay datos');
END //
DELIMITER ;
```

```
DELIMITER //
CREATE FUNCTION fn_ingredientes_caducando(
    p_dias INT
) RETURNS INT
READS SQL DATA
BEGIN
    DECLARE v_count INT;

    SELECT COUNT(*) INTO v_count
    FROM ingredientes
    WHERE caduca = TRUE
    AND dias_caducidad <= p_dias
    AND stock_actual > 0;

    RETURN v_count;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE FUNCTION fn_calcular_propina(
    p_pedido_id INT,
    p_porcentaje DECIMAL(5,2)
) RETURNS DECIMAL(10,2)
READS SQL DATA
BEGIN
    DECLARE v_total DECIMAL(10,2);

    SELECT total INTO v_total FROM pedidos WHERE pedido_id = p_pedido_id;

    RETURN ROUND(v_total * (p_porcentaje / 100), 2);
END //
DELIMITER ;
```

13.Ejemplo de Triggers aplicados al proyecto formativo.

```
DELIMITER //
CREATE TRIGGER tr_actualizar_stock_after_insert_detalle
AFTER INSERT ON detalles_pedido
FOR EACH ROW
BEGIN
    DECLARE v_ingrediente_id INT;
    DECLARE v_cantidad_necesaria DECIMAL(10,2);
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT ingrediente_id, cantidad
        FROM platillo_ingredientes
        WHERE platillo_id = NEW.platillo_id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO v_ingrediente_id, v_cantidad_necesaria;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Actualizar stock (restar)
        UPDATE ingredientes
        SET stock_actual = stock_actual - (v_cantidad_necesaria * NEW.cantidad)
        WHERE ingrediente_id = v_ingrediente_id;
    END LOOP;
    CLOSE cur;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER tr_verificar_stock_minimo
AFTER UPDATE ON ingredientes
FOR EACH ROW
BEGIN
    IF NEW.stock_actual < NEW.stock_minimo THEN
        INSERT INTO inventario (
            ingrediente_id,
            usuario_id,
            cantidad,
            tipo_movimiento,
            motivo,
            costo_total
        )
        VALUES (
            NEW.ingrediente_id,
            1, -- Usuario sistema
            NEW.stock_minimo * 2, -- Cantidad sugerida para reabastecer
            'entrada',
            'Reabastecimiento automático por stock mínimo',
            NEW.costo_por_unidad * (NEW.stock_minimo * 2)
        );
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER tr_log_cambios_pedidos
AFTER UPDATE ON pedidos
FOR EACH ROW
BEGIN
    IF OLD.estado != NEW.estado THEN
        INSERT INTO logs_pedidos (
            pedido_id,
            estado_anterior,
            estado_nuevo,
            fecha_cambio
        )
        VALUES (
            NEW.pedido_id,
            OLD.estado,
            NEW.estado,
            NOW()
        );
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER tr_validar_email_usuario
BEFORE INSERT ON usuarios
FOR EACH ROW
BEGIN
    IF NEW.email NOT LIKE '%@%.%' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El formato del email no es válido';
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER tr_actualizar_disponibilidad_platillo
AFTER UPDATE ON ingredientes
FOR EACH ROW
BEGIN
    DECLARE v_platillo_id INT;
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR
        SELECT DISTINCT platillo_id
        FROM platillo_ingredientes
        WHERE ingrediente_id = NEW.ingrediente_id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO v_platillo_id;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Verificar disponibilidad de todos los ingredientes
        IF NOT fn_verificar_disponibilidad_ingrediente(v_platillo_id, 1) THEN
            UPDATE platillos SET activo = FALSE WHERE platillo_id = v_platillo_id;
        ELSE
            UPDATE platillos SET activo = TRUE WHERE platillo_id = v_platillo_id;
        END IF;
    END LOOP;
    CLOSE cur;
END //
DELIMITER ;
```

14. Formas de encriptación de contraseña en SQL.