**Introducción**

Este proyecto consiste en el diseño de una base de datos para una repostera, con el objetivo de organizar y gestionar eficientemente la información relacionada con productos, pedidos, ingredientes y clientes. El enfoque del modelo es práctico y apunta a simplificar el manejo diario de datos clave como recetas, ventas y stock.

**Objetivo**

El objetivo principal del proyecto es implementar una estructura de base de datos relacional que permita a la repostera registrar sus pedidos, administrar los ingredientes utilizados en cada receta y obtener un mayor control sobre sus productos y clientes. Esto también posibilita gestionar de forma ordenada los aspectos logísticos del negocio, como la fecha de entrega de pedidos, el seguimiento de productos vendidos y los costos por unidad.

**Situación Problemática**

La repostera necesita un sistema que le permita dejar de depender de registros manuales, mensajes o planillas desorganizadas, y centralizar toda la información crítica de su negocio. Actualmente, la falta de una estructura ordenada provoca errores en los pedidos, pérdida de información sobre ingredientes, dificultades para calcular cantidades o precios, y complicaciones al momento de gestionar los clientes o hacer análisis básicos. La implementación de esta base de datos busca cubrir esas brechas, permitiendo una administración clara, escalable y confiable de todos los procesos clave del emprendimiento.

**Modelo de Negocio**

La organización que se representa en este proyecto es un emprendimiento individual de repostería. Su modelo de negocio se centra en la producción y venta personalizada de productos como tortas, cupcakes y otros postres, preparados por pedido.

Los pedidos pueden tener particularidades como decoraciones especiales, cantidades personalizadas o ingredientes adicionales, lo cual hace necesario un sistema flexible y detallado para registrar cada transacción.

El negocio opera bajo demanda, es decir, produce solo lo que vende, por lo que resulta fundamental controlar con precisión los ingredientes usados por receta, los costos asociados, la planificación de entregas y la gestión de los clientes. La solución propuesta permite gestionar todos estos aspectos desde una base de datos relacional estructurada.

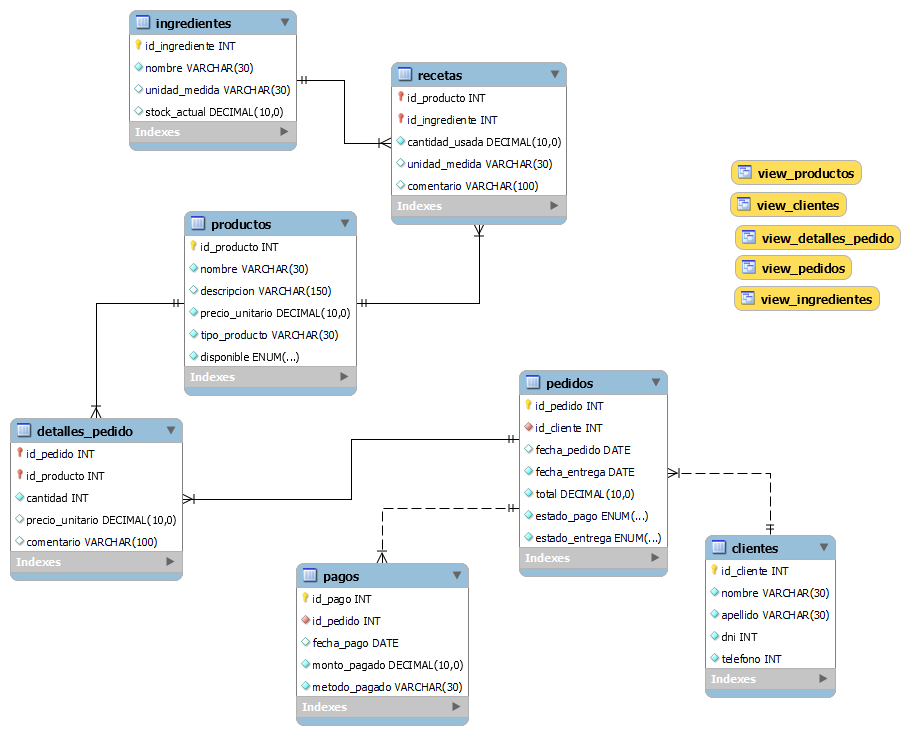
**Diagrama Entidad-Relación (E-R)**

El diagrama E-R representa de manera fiel la estructura de la base de datos generada a partir del script SQL del proyecto Tortas Mariluz. Incluye todas las entidades del sistema: productos, clientes, pedidos, detalles\_pedido, pagos, ingredientes y recetas.

Se muestran también las relaciones entre ellas, destacando:

* La relación entre clientes y pedidos, donde un cliente puede tener múltiples pedidos.
* La tabla intermedia detalles\_pedido, que vincula pedidos con productos, permitiendo registrar múltiples productos por pedido, su cantidad, precio y un comentario adicional personalizado.
* La relación de pedidos con pagos, permitiendo registrar diferentes métodos y fechas de pago.
* La relación entre productos e ingredientes, modelada mediante la tabla intermedia recetas, que especifica qué ingredientes y en qué cantidades se utilizan para cada producto.

Diagrama:



**Listado de Tablas:**

CREATE TABLE IF NOT EXISTS productos (

    id\_producto INT AUTO\_INCREMENT PRIMARY KEY,

    nombre VARCHAR(30) NOT NULL,

    descripcion VARCHAR(150) NULL DEFAULT "Sin Descripcion",

    precio\_unitario DECIMAL NOT NULL,

    tipo\_producto VARCHAR(30) NOT NULL,  *-- "Torta", "cupcake", "tarta", etc*

    disponible ENUM("Disponible", "No Disponible") NOT NULL

);

CREATE TABLE IF NOT EXISTS clientes (

    id\_cliente INT AUTO\_INCREMENT PRIMARY KEY,

    nombre VARCHAR(30) NOT NULL,

    apellido VARCHAR(30) NOT NULL,

    dni INT NOT NULL,

    telefono INT NOT NULL

);

CREATE TABLE IF NOT EXISTS pedidos (

    id\_pedido INT AUTO\_INCREMENT PRIMARY KEY,

    id\_cliente INT NOT NULL,

    fecha\_pedido DATE,         *-- tiene CURRENT\_DATE*

    fecha\_entrega DATE NOT NULL,

    total DECIMAL NOT NULL,

    estado\_pago ENUM('Pendiente', 'Pagado', 'Parcial') NOT NULL,

    estado\_entrega ENUM('Pendiente', 'Entregado', 'Cancelado') NOT NULL,

    FOREIGN key (id\_cliente) REFERENCES clientes(id\_cliente)

);

*/\* Tabla intermedia entre pedidos y productos \*/*

CREATE TABLE IF NOT EXISTS detalles\_pedido (

    id\_pedido INT NOT NULL,

    id\_producto INT NOT NULL,

    cantidad INT NOT NULL,

    precio\_unitario DECIMAL,

    comentario VARCHAR(100) NULL DEFAULT "Sin Comentario",    *-- si desea algun extra en el producto (decoracion)*

    PRIMARY KEY (id\_pedido, id\_producto),

    FOREIGN KEY (id\_pedido) REFERENCES pedidos(id\_pedido),

    FOREIGN KEY (id\_producto) REFERENCES productos(id\_producto)

);

CREATE TABLE IF NOT EXISTS pagos (

    id\_pago INT AUTO\_INCREMENT PRIMARY key,

    id\_pedido INT NOT NULL,

    fecha\_pago DATE,        *-- tiene CURRENT\_DATE*

    monto\_pagado DECIMAL NOT NULL,

    metodo\_pagado VARCHAR(30) NOT NULL,

    FOREIGN key (id\_pedido) REFERENCES pedidos(id\_pedido)

);

CREATE TABLE IF NOT EXISTS ingredientes (

    id\_ingrediente INT AUTO\_INCREMENT PRIMARY key,

    nombre VARCHAR(30) NOT NULL,

    unidad\_medida VARCHAR(30) NULL,

    stock\_actual DECIMAL NULL

);

*/\* Tabla intermedia entre productos y ingredientes \*/*

CREATE TABLE IF NOT EXISTS recetas (

    id\_producto INT NOT NULL,

    id\_ingrediente INT NOT NULL,

    cantidad\_usada DECIMAL NOT NULL,

    unidad\_medida VARCHAR(30) NULL,

    comentario VARCHAR(100) NULL DEFAULT "Sin Comentarios",    *-- Indicar como se dividen las cantidades en caso de usar el mismo ingrediente dos veces en la receta*

    PRIMARY KEY (id\_producto, id\_ingrediente),

    FOREIGN KEY (id\_producto) REFERENCES productos(id\_producto),

    FOREIGN KEY (id\_ingrediente) REFERENCES ingredientes(id\_ingrediente)

);

**Vistas:**

1. CREATE VIEW view\_productos AS SELECT id\_producto, nombre, tipo\_producto, disponible FROM tortas\_mariluz.productos;

SELECT \* FROM view\_productos ;

Se visualiza el id del producto el nombre y el tipo de producto, el objetivo consta de tener una vista del nombre y tipo del producto. Proviene de la tabla ‘productos’

**2.** CREATE VIEW view\_clientes AS SELECT nombre, apellido, telefono FROM tortas\_mariluz.clientes;

SELECT \* FROM view\_clientes;

Se visualiza el nombre apellido y teléfono del cliente, con el objetivo de solo tener datos esenciales del cliente. Proviene de la tabla ‘clientes’

**3.** CREATE VIEW view\_pedidos AS SELECT id\_pedido, id\_cliente, total FROM tortas\_mariluz.pedidos ;

SELECT \* FROM view\_pedidos;

Se visualiza el id del pedido y el id del cliente, el objetivo saber de qué producto y cliente se trata se trata. Proviene de la tabla ‘pedidos’

**4.** CREATE VIEW view\_ingredientes AS SELECT nombre, stock\_actual FROM tortas\_mariluz.ingredientes ;

SELECT \* FROM view\_ingredientes;

Se visualiza el nombre del ingrediente y el stock, el objetivo saber si cuenta con un stock adecuado. Proviene de la tabla ingredientes

**5.** CREATE VIEW view\_detalles\_pedido AS SELECT id\_pedido, id\_producto, cantidad, comentario FROM tortas\_mariluz.detalles\_pedido ;

SELECT \* FROM view\_detalles\_pedido;

Se visualiza el id del pedido, el id del producto cantidad del producto y un comentario, el objetivo ver la cantidad de productos que tiene el pedido

**Funciones:**

**Funcion 1 –** stock\_disponible\_ingrediente

Obtenemos el stock actual del ingrediente del id que se pasa por parámetro. El objetivo es obtener solo el stock del producto y no otros valores del mismo.

DELIMITER $$

CREATE FUNCTION stock\_disponible\_ingrediente(

    p\_id\_ingrediente INT

)

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

    DECLARE v\_stock DECIMAL(10,2);

*-- Obtener stock del ingrediente*

    SELECT stock\_actual

    INTO v\_stock

    FROM ingredientes

    WHERE id\_ingrediente = p\_id\_ingrediente;

    RETURN v\_stock;

END$$

DELIMITER ;

*-- Usar Funciones*

*/\* El as para que cuando muestre la query se muestre stock\_oreo y no stock\_disponible\_ingrediente(7) \*/*

SELECT stock\_disponible\_ingrediente(7) AS stock\_oreo;

**funcion 2 –** calcular\_total\_pedido

Nos retorna el monto total del pedido, asi podemos conseguir de forma rápida el monto total de un pedido.

DELIMITER $$

CREATE FUNCTION calcular\_total\_pedido(

    p\_id\_pedido INT

)

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

    DECLARE v\_total DECIMAL(10,2);

*-- Calcular total sumando cantidad \* precio\_unitario de cada producto del pedido*

    SELECT SUM(cantidad \* precio\_unitario)

    INTO v\_total

    FROM detalles\_pedido

    WHERE id\_pedido = p\_id\_pedido;

    RETURN v\_total;

END$$

DELIMITER ;

*-- Usar Funciones*

SELECT calcular\_total\_pedido(3) AS total\_pedido;

**Stored Procedures:**

**SP1 –** registrar\_pago

Crea un registro en la tabla pagos.

Se pasa por parámetro el ‘id del pedido’ el ‘monto pagado’ y el ‘metodo de pago’. Luego inserta el pago, calcula el monto que el cliente a abanado y lo guarda en la variable v\_pagado. Trae el monto total de la tabla pedidos y actualiza el estado del pedido dependiendo del monto que haya sido abonado, estos pueden ser ‘Pagado’, ‘Parcial’ o ‘Pendiente’

El objetivo es crear los pagos de forma rápida y sencilla sin tener que viajar entre tablas lo cual puede crear confusiones.

DELIMITER $$

CREATE PROCEDURE registrar\_pago (

    IN p\_id\_pedido INT,

    IN p\_monto DECIMAL(10,2),

    IN p\_metodo VARCHAR(30)

)

BEGIN

    DECLARE v\_total DECIMAL(10,2);

    DECLARE v\_pagado DECIMAL(10,2);

*-- Insertar el pago*

    INSERT INTO pagos (id\_pedido, fecha\_pago, monto\_pagado, metodo\_pagado)

    VALUES (p\_id\_pedido, CURRENT\_DATE, p\_monto, p\_metodo);

*-- Calcular cuánto lleva pagado*

    SELECT IFNULL(SUM(monto\_pagado),0) INTO v\_pagado

    FROM pagos

    WHERE id\_pedido = p\_id\_pedido;

*-- Traer total del pedido*

    SELECT total INTO v\_total

    FROM pedidos

    WHERE id\_pedido = p\_id\_pedido;

*-- Actualizar estado*

    IF v\_pagado >= v\_total THEN

        UPDATE pedidos SET estado\_pago = 'Pagado'

        WHERE id\_pedido = p\_id\_pedido;

    ELSEIF v\_pagado > 0 THEN

        UPDATE pedidos SET estado\_pago = 'Parcial'

        WHERE id\_pedido = p\_id\_pedido;

    ELSE

        UPDATE pedidos SET estado\_pago = 'Pendiente'

        WHERE id\_pedido = p\_id\_pedido;

    END IF;

END$$

DELIMITER ;

*-- Usar Stored Producers*

*/\* Agrego nuevo pedido para probar el SP \*/*

INSERT INTO pedidos (id\_cliente, fecha\_entrega, total, estado\_pago, estado\_entrega) VALUES (1, '2025-09-06', 25000, "pendiente", "Pendiente");

CALL registrar\_pago(4, 25000, 'Mercado Pago');

**SP 2 –** actualizar\_stock

Actualiza el stock al tener un nuevo pedido en la tabla pedidos.

Requiere de parámetro el id del pedido, usamos un cursor para recorres los productos del pedido, usamos la tabla intermedia ‘detalles\_pedido’ para tener todos los productos del pedido. En cada loop resta todos los ingredientes que conlleve la receta del producto y verifica el comentario de la tabla ‘detalles\_pedido’ para restar este otro valor de la columna.

Con esto se puede mejorar el sistema de stock al tener un nuevo pedido en el sistema.

DELIMITER $$

CREATE PROCEDURE actualizar\_stock (

    IN p\_id\_pedido INT

)

BEGIN

    DECLARE done INT DEFAULT 0;

    DECLARE v\_id\_producto INT;

    DECLARE v\_cantidad\_producto DECIMAL(10,2);

    DECLARE v\_comentario VARCHAR(100);

*-- Cursor para recorrer los productos del pedido, usamos la tabla intermedia detalles\_pedido*

    DECLARE cur\_productos CURSOR FOR

        SELECT id\_producto, cantidad, comentario

        FROM detalles\_pedido

        WHERE id\_pedido = p\_id\_pedido;

*/\* Cuando el cursor llega al final de las filas, MySQL genera un error NOT FOUND.*

*Este handler evita que eso detenga el SP.*

*En lugar de error, pone done = 1 para indicar que no quedan más productos por recorrer. \*/*

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

*-- Activa el cursor para empezar a recorrer las filas.*

    OPEN cur\_productos;

*/\* FETCH cur\_productos INTO ... → toma la siguiente fila del cursor y guarda los valores en las variables \*/*

*-- leer\_productos es el nombre del loop*

    leer\_productos: LOOP

        FETCH cur\_productos INTO v\_id\_producto, v\_cantidad\_producto, v\_comentario;

        IF done = 1 THEN

            LEAVE leer\_productos;

        END IF;

*-- Resta todos los ingredientes del producto según receta*

        UPDATE ingredientes i

        JOIN recetas r ON i.id\_ingrediente = r.id\_ingrediente

        SET i.stock\_actual = i.stock\_actual - (r.cantidad\_usada \* v\_cantidad\_producto)

        WHERE r.id\_producto = v\_id\_producto;

*-- Si el comentario contiene "Oreos de decoracion", restar Oreos según receta*

        IF v\_comentario LIKE '%Oreos de decoracion%' THEN

            UPDATE ingredientes i

            JOIN recetas r ON i.id\_ingrediente = r.id\_ingrediente

            SET i.stock\_actual = i.stock\_actual - (r.cantidad\_usada \* v\_cantidad\_producto)

            WHERE r.id\_producto = v\_id\_producto AND i.nombre = 'oreos';

        END IF;

*-- Si dice 'Sin Comentario', no hace nada*

    END LOOP;

    CLOSE cur\_productos;

END$$

DELIMITER ;

*-- Usar Stored Producers*

*/\* Agrego el detalle pedido del nuevo pedido \*/*

INSERT INTO detalles\_pedido (id\_pedido, id\_producto, cantidad, precio\_unitario, comentario) VALUES (4, 3, 1, 25000, "Oreos de decoracion");

*/\* Insertamos la receta del producto \*/*

INSERT INTO recetas (id\_producto, id\_ingrediente, cantidad\_usada, unidad\_medida) VALUES (3, 7, 8, "unidad");

INSERT INTO recetas (id\_producto, id\_ingrediente, cantidad\_usada, unidad\_medida) VALUES (3, 2, 700, "gramos");

INSERT INTO recetas (id\_producto, id\_ingrediente, cantidad\_usada, unidad\_medida) VALUES (3, 3, 6, "unidad");

INSERT INTO recetas (id\_producto, id\_ingrediente, cantidad\_usada, unidad\_medida) VALUES (3, 4, 500, "mililitros");

CALL actualizar\_stock(4);

**Triggers:**

DELIMITER $$

*-- Cuando se inserta un detalle*

CREATE TRIGGER trg\_detalles\_pedido\_insert

AFTER INSERT ON detalles\_pedido

FOR EACH ROW

BEGIN

    UPDATE pedidos

    SET total = total + (NEW.cantidad \* NEW.precio\_unitario)

    WHERE id\_pedido = NEW.id\_pedido;

END$$

*-- Cuando se actualiza un detalle (ej: cambia cantidad o precio)*

CREATE TRIGGER trg\_detalles\_pedido\_update

AFTER UPDATE ON detalles\_pedido

FOR EACH ROW

BEGIN

    UPDATE pedidos

    SET total = total - (OLD.cantidad \* OLD.precio\_unitario) + (NEW.cantidad \* NEW.precio\_unitario)

    WHERE id\_pedido = NEW.id\_pedido;

END$$

*-- Cuando se elimina un detalle*

CREATE TRIGGER trg\_detalles\_pedido\_delete

AFTER DELETE ON detalles\_pedido

FOR EACH ROW

BEGIN

    UPDATE pedidos

    SET total = total - (OLD.cantidad \* OLD.precio\_unitario)

    WHERE id\_pedido = OLD.id\_pedido;

END$$

DELIMITER ;

**Ejemplo de uso:**

**1.** Crear un pedido vacío:

INSERT INTO pedidos (id\_cliente, fecha\_pedido, fecha\_entrega, total, estado\_pago, estado\_entrega)

VALUES (1, CURRENT\_DATE, '2025-09-10', 0, 'Pendiente', 'Pendiente');

**Supongamos que este pedido tiene id\_pedido = 1.**

**2.** Insertar detalles:

INSERT INTO detalles\_pedido (id\_pedido, id\_producto, cantidad, precio\_unitario)

VALUES (1, 2, 3, 150.00); /\*\* Producto 2, cantidad 3, precio 150 \*\*/

Automáticamente el trigger suma 3 \* 150 = 450 al total en pedidos.

**3.** Si actualizás la cantidad:

UPDATE detalles\_pedido

SET cantidad = 5

WHERE id\_pedido = 1 AND id\_producto = 2;

El trigger recalcula el total (resta 3\*150, suma 5\*150).

**4.** Si eliminás el detalle:

DELETE FROM detalles\_pedido

WHERE id\_pedido = 1 AND id\_producto = 2;

El trigger descuenta el subtotal de ese producto del total.

Con esto, **el campo total en ‘pedidos’ siempre queda consistente**.