UNIVERSIDAD METROPOLITANA DE HONDURAS

Innovación, Valores, Liderazgo

Universidad Metropolitana de Honduras

Diseño de base de datos 2

Docente: Allan Lopez

Informe del Proyecto Final

ALUMNOS:

Andres Alessandro Viera Espinal

Carlos Roberto Diaz

Kilvet Alonso Barahona Martinez

Dagoberto Fernandez Valenzuela

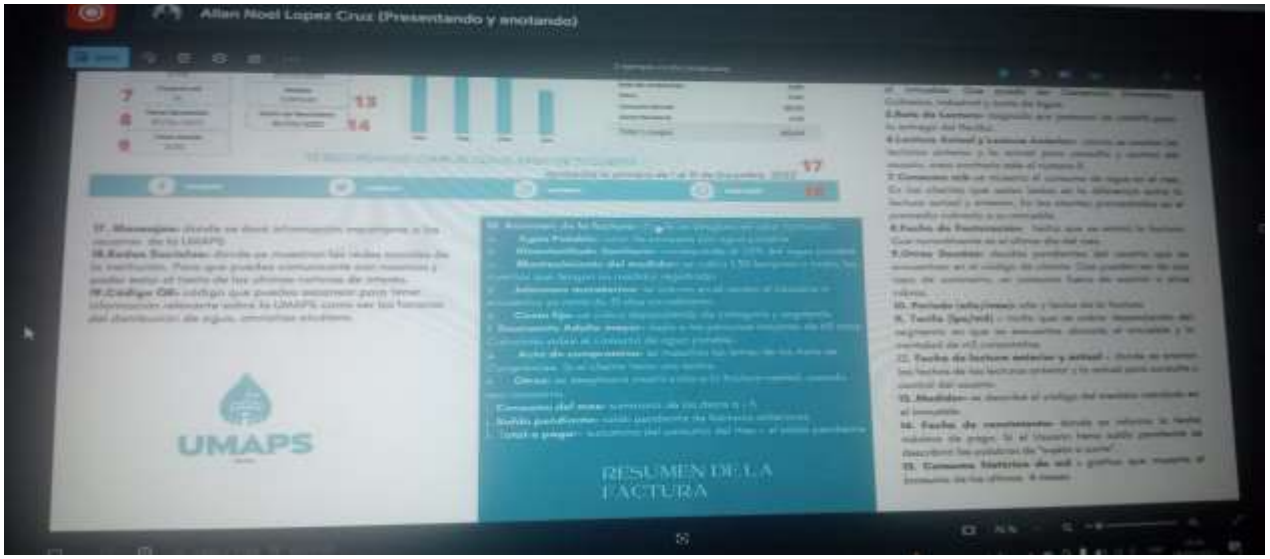
Proyecto Creacion de Fomulario UMAPS

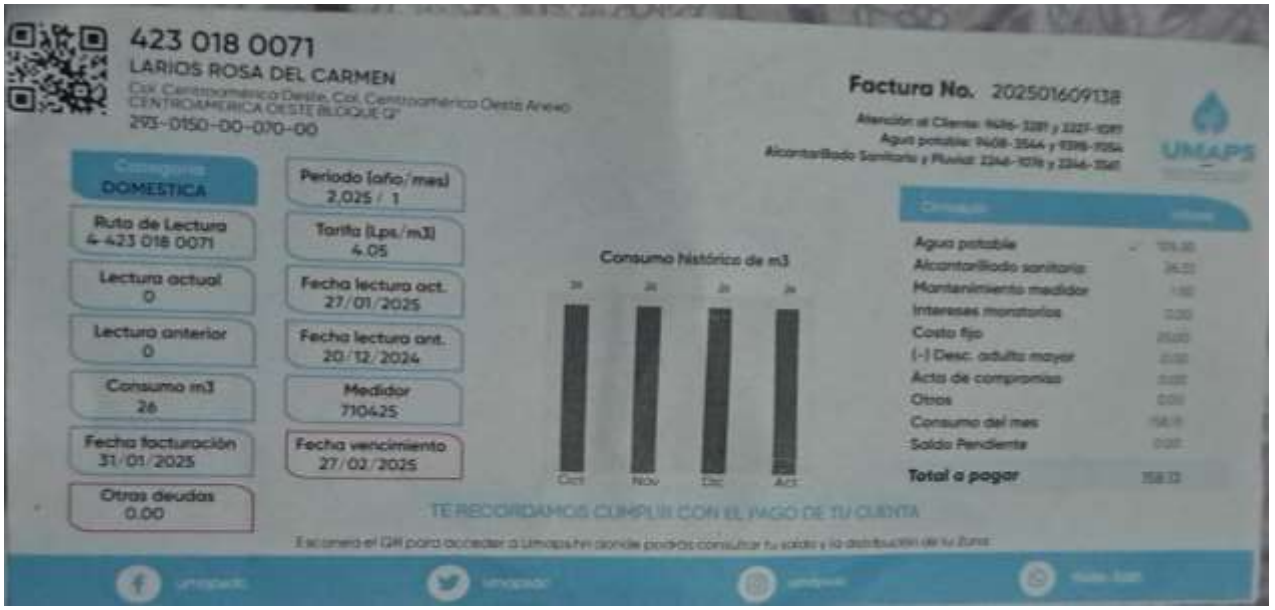
Descripción del problema

Para transformar este formulario en una base de datos relacional, identificamos las principales entidades y sus atributos. Un modelo relacional nos permite organizar la información en tablas con relaciones claras, evitando redundancias y mejorando la gestión de datos.

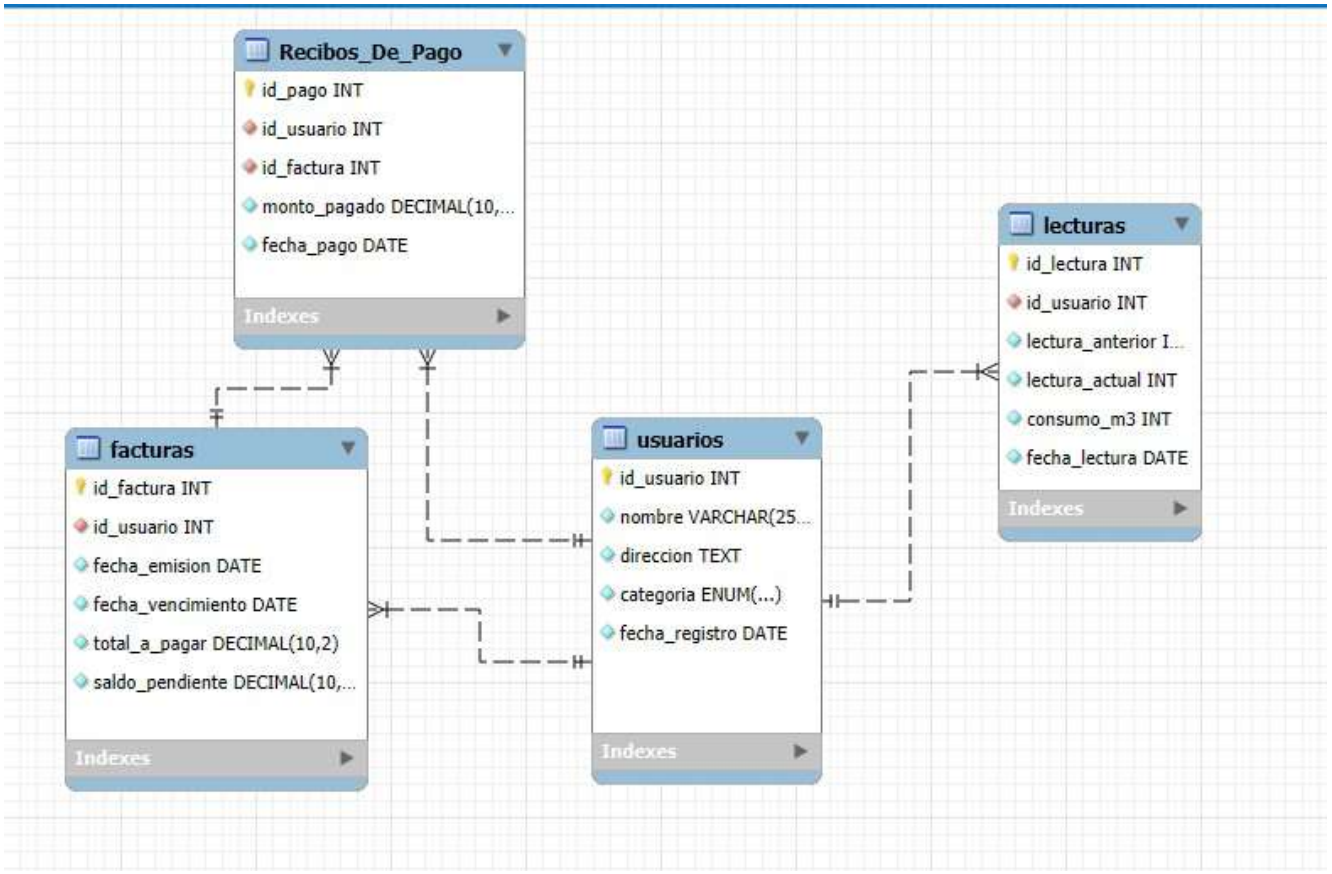
1. **Usuarios**: Identifica a los clientes de UMAPS con datos como nombre, dirección y tipo de usuario.
2. **Facturas**: Registra los detalles de facturación de cada usuario, incluyendo fechas, montos y estado de pago.
3. **Recibos de Pagos**: Lleva el control de los pagos realizados y sus métodos.
4. **Lecturas de Medidor**: Guarda las mediciones de consumo de agua registradas periódicamente.

Con esta estructura, podemos diseñar un **modelo entidad-relación (ER)** y luego convertirlo en tablas SQL normalizadas para gestionar eficientemente la información.





Modelo relacional



Script DDL de base de datos

Creación de base de datos

```
/*
Codigo de creación
Se crea una base de datos para guardar nuestras tablas
Se crean las diferentes tablas basadas en nuestro modelo
*/

DROP DATABASE IF EXISTS sistema_facturacion_agua;
CREATE DATABASE sistema_facturacion_agua;
USE sistema_facturacion_agua;

-- Tabla de Usuarios
CREATE TABLE usuarios (
    id_usuario INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    direccion TEXT NOT NULL,
    categoria ENUM('Residencial', 'Comercial', 'Industrial') NOT NULL,
    fecha_registro DATE NOT NULL
);

-- Tabla de Lecturas de Consumo de Agua
CREATE TABLE lecturas (
    id_lectura INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
    lectura_anterior INT NOT NULL,
    lectura_actual INT NOT NULL,
    consumo_m3 INT GENERATED ALWAYS AS (lectura_actual - lectura_anterior) STORED,
    fecha_lectura DATE NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
);

-- Tabla de Facturas
CREATE TABLE facturas (
    id_factura INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
    fecha_emision DATE NOT NULL,
    fecha_vencimiento DATE NOT NULL,
    total_a_pagar DECIMAL(10,2) NOT NULL,
    saldo_pendiente DECIMAL(10,2) NOT NULL DEFAULT 0,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
);
```

```
-- Nueva Tabla de Conceptos
CREATE TABLE conceptos (
    id_concepto INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL UNIQUE,
    descripcion TEXT
);


-- Tabla de Recibos de Pago
CREATE TABLE recibos_de_pago (
    id_pago INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
    id_factura INT NOT NULL,
    monto_pagado DECIMAL(10,2) NOT NULL,
    fecha_pago DATE NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_factura) REFERENCES facturas(id_factura)
);



-- Nueva Tabla de Detalle de Factura para asociar conceptos con facturas
CREATE TABLE detalle_factura (
    id_detalle INT AUTO_INCREMENT PRIMARY KEY,
    id_factura INT NOT NULL,
    id_concepto INT NOT NULL,
    monto DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (id_factura) REFERENCES facturas(id_factura) ON DELETE CASCADE,
    FOREIGN KEY (id_concepto) REFERENCES conceptos(id_concepto) ON DELETE CASCADE
);

select * from sistema_facturacion_agua.conceptos;
```

Script DML de datos de prueba

Inserción de datos de prueba

```
/*
Datos de prueba
Codigo para insertar diferentes datos en las tablas ya existentes para realizar procedimientos
*/
USE sistema_facturacion_agua;
-- Insertar usuarios
INSERT INTO usuarios (nombre, direccion, categoria, fecha_registro) VALUES
('Juan Pérez', 'Av. Principal #123, Ciudad', 'Residencial', '2024-01-10'),
('María López', 'Calle Secundaria #456, Ciudad', 'Comercial', '2023-11-25'),
('Carlos Ramírez', 'Colonia Centro #789, Ciudad', 'Residencial', '2022-07-05');

-- Insertar lecturas de consumo de agua
INSERT INTO lecturas (id_usuario, lectura_anterior, lectura_actual, fecha_lectura) VALUES
(1, 100, 120, '2024-03-01'), -- Consumo: 20 m3
(2, 250, 270, '2024-03-01'), -- Consumo: 20 m3
(3, 500, 530, '2024-03-01'); -- Consumo: 30 m3

-- Insertar facturas de los usuarios
INSERT INTO facturas (id_usuario, fecha_emision, fecha_vencimiento, total_a_pagar,
saldo_pendiente) VALUES
(1, '2024-03-05', '2024-03-20', 300.50, 0.00),
(2, '2024-03-05', '2024-03-20', 450.75, 50.00), -- Saldo pendiente de meses anteriores
(3, '2024-03-05', '2024-03-20', 600.25, 20.00);
INSERT INTO conceptos (nombre, descripcion) VALUES
('Agua potable', 'Cobro por el consumo de agua potable'),
('Alcantarillado sanitario', 'Servicio de alcantarillado'),
('Mantenimiento medidor', 'Costo de mantenimiento del medidor'),
('Intereses moratorios', 'Intereses por mora en el pago'),
('Costo fijo', 'Tarifa fija mensual'),
('(-) Desc. adulto mayor', 'Descuento para adultos mayores'),
('Acta de compromiso', 'Pago de acuerdo a acta de compromiso'),
('Otros', 'Otros cobros adicionales'),
('Consumo del mes', 'Cargo por el consumo del mes'),
('Saldo Pendiente', 'Monto no pagado de periodos anteriores')
ON DUPLICATE KEY UPDATE descripcion = VALUES(descripcion);
```

```
-- Insertar detalles de facturas con conceptos
INSERT INTO detalle_factura (id_factura, id_concepto, monto) VALUES
(1, 1, 150.00), -- Agua potable
(1, 2, 50.00), -- Alcantarillado sanitario
(1, 5, 100.50), -- Costo fijo

(2, 1, 200.00), -- Agua potable
(2, 2, 75.00), -- Alcantarillado sanitario
(2, 4, 25.75), -- Intereses moratorios
(2, 5, 100.00), -- Costo fijo

(3, 1, 300.00), -- Agua potable
(3, 2, 100.00), -- Alcantarillado sanitario
(3, 5, 150.25); -- Costo fijo

-- Insertar pagos de usuarios
INSERT INTO recibos_de_pago (id_usuario, id_factura, monto_pagado, fecha_pago) VALUES
(1, 1, 300.50, '2024-03-10'), -- Pago total
(2, 2, 400.00, '2024-03-15'), -- Pago parcial
(3, 3, 600.25, '2024-03-18'); -- Pago total

/*
Mostrar Datos insertados
en las tablas
por medio de un select from que mostrara la información registrada
*/

select * from sistema_facturacion_agua.usuarios;

select * from sistema_facturacion_agua.lecturas;

select * from sistema_facturacion_agua.detalle_factura;

select * from sistema_facturacion_agua.facturas;

select * from sistema_facturacion_agua.conceptos;
```

Script DDL de procedimientos

Creación de procedimiento almacenado

```
/*Proceso 1       ResumenAnual*/
DELIMITER $$
/* ejecutar el drop si se desea realizar el procedimiento por segunda vez
DROP PROCEDURE IF EXISTS ResumenAnual;
*/
CREATE PROCEDURE ResumenAnual(
   IN p_anio INT
)
BEGIN
   SELECT
      u.id_usuario,
      u.nombre AS Nombre,
      p_anio AS Anio,
      COALESCE(SUM(l.consumo_m3), 0) AS Consumo_Total_m3,
      COALESCE(SUM(f.total_a_pagar), 0) AS Total_Facturado,
      COALESCE(SUM(r.monto_pagado), 0) AS Total_Pagado
   FROM usuarios u
   LEFT JOIN lecturas l ON u.id_usuario = l.id_usuario AND YEAR(l.fecha_lectura) = p_anio
   LEFT JOIN facturas f ON u.id_usuario = f.id_usuario AND YEAR(f.fecha_emision) = p_anio
   LEFT JOIN recibos_de_pago r ON u.id_usuario = r.id_usuario AND YEAR(r.fecha_pago) =
p_anio
   GROUP BY u.id_usuario, u.nombre
   ORDER BY Total_Facturado DESC;
END $$

DELIMITER ;

CALL ResumenAnual(2025);
```

```
/* procedimiento 2 ListarUsuariosPorCategoriaYAnio */
DELIMITER $$
/* ejecutar el drop si se desea realizar el procedimiento por segunda vez
DROP PROCEDURE IF EXISTS ListarUsuariosPorCategoriaYAnio;
*/
CREATE PROCEDURE ListarUsuariosPorCategoriaYAnio(
    IN p_categoria ENUM('Residencial', 'Comercial', 'Industrial'),
    IN p_anio INT
)
BEGIN
    SELECT
        id_usuario,
        nombre,
        direccion,
        categoria,
        YEAR(fecha_registro) AS anio_registro
    FROM usuarios
    WHERE categoria = p_categoria
      AND YEAR(fecha_registro) = p_anio;
END $$

DELIMITER ;

CALL ListarUsuariosPorCategoriaYAnio('Residencial', 2024);
```

```
/*Procedimiento 3*/
```

```
DELIMITER $$

/* ejecutar el drop si se desea realizar el procedimiento por segunda vez
DROP PROCEDURE IF EXISTS montosporaños;
*/
CREATE PROCEDURE montosporaños(
    IN p_anio INT
)
BEGIN

    SELECT
        u.nombre AS nombre_usuario,
        f.total_a_pagar,
        YEAR(f.fecha_emision) AS anio_factura
    FROM usuarios u
    JOIN facturas f ON u.id_usuario = f.id_usuario
    WHERE YEAR(f.fecha_emision) = p_anio;
END $$

DELIMITER ;

CALL montosporaños(2024);
```

Script DDL de funciones

Creación de funciones almacenadas

/*Función para calcular el total a pagar de una factura, tomando en cuenta los pagos parciales y el saldo pendiente*/

```
DELIMITER $$

/* ejecutar el drop si se desea realizar el procedimiento por segunda vez
DROP FUNCTION IF EXISTS calcular_total_a_pagar;
*/
CREATE FUNCTION calcular_total_a_pagar(id_factura INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
  DECLARE total DECIMAL(10,2);
  DECLARE saldo DECIMAL(10,2);
  DECLARE total_pagado DECIMAL(10,2);

  -- Obtener el total de la factura (LIMIT 1 para evitar más de una fila)
  SELECT total_a_pagar INTO total
  FROM facturas
  WHERE id_factura = id_factura
  LIMIT 1;

  -- Obtener el saldo pendiente (LIMIT 1 para evitar más de una fila)
  SELECT saldo_pendiente INTO saldo
  FROM facturas
  WHERE id_factura = id_factura
  LIMIT 1;

  -- Obtener el total pagado hasta el momento
  SELECT SUM(monto_pagado) INTO total_pagado
  FROM recibos_de_pago
  WHERE id_factura = id_factura;
```

```
    -- Calcular el total a pagar considerando el saldo pendiente y los pagos realizados
    RETURN total - total_pagado + saldo;
END $$

DELIMITER ;SELECT calcular_total_a_pagar(1);  -- Donde 1 es el ID de la factura.
/* Función para obtener el consumo total de agua de un usuario específico entre dos fechas */


DELIMITER $$
/* ejecutar el drop si se desea realizar el procedimiento por segunda vez
DROP FUNCTION IF EXISTS obtener_consumo_usuario;
*/
CREATE FUNCTION obtener_consumo_usuario(id_usuario INT, fecha_inicio DATE, fecha_fin
DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE total_consumo INT;

    -- Sumar el consumo de agua entre las lecturas de las fechas proporcionadas
    SELECT SUM(consumo_m3) INTO total_consumo
    FROM lecturas
    WHERE id_usuario = id_usuario
    AND fecha_lectura BETWEEN fecha_inicio AND fecha_fin;

    RETURN total_consumo;
END $$

DELIMITER ;

SELECT obtener_consumo_usuario(1, '2024-01-01', '2024-03-01');  -- Donde 1 es el ID del usuario
```

```
/* Función para obtener el descuento aplicado a un usuario (en caso de tener descuento por adulto mayor o algún otro concepto)*/

DELIMITER $$
/* ejecutar el drop si se desea realizar el procedimiento por segunda vez
DROP FUNCTION IF EXISTS obtener_descuento_usuario;
*/
CREATE FUNCTION obtener_descuento_usuario(id_usuario INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE descuento DECIMAL(10,2);

    -- Sumar los descuentos aplicados al usuario desde el detalle de la factura
    SELECT SUM(monto) INTO descuento
    FROM detalle_factura df
    JOIN conceptos c ON df.id_concepto = c.id_concepto
    WHERE df.id_factura IN (SELECT id_factura FROM facturas WHERE id_usuario = id_usuario)
    AND c.nombre LIKE '%Desc.%'; -- Asumiendo que los descuentos tienen "Desc." en el nombre del concepto

    IF descuento IS NULL THEN
        SET descuento = 0;
    END IF;

    RETURN descuento;
END $$

DELIMITER ;


SELECT obtener_descuento_usuario(1);  -- Donde 1 es el ID del usuario
```

Script DDL triggers

Creación de triggers

```
-- SELECCIONAR O CREAR BASE DE DATOS
CREATE DATABASE IF NOT EXISTS sistema_facturacion;
USE sistema_facturacion;
-- CREACIÓN DE TABLA RECIBOS DE PAGO
CREATE TABLE Recibos_De_Pago (
    id_pago INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT NOT NULL,
    id_factura INT NOT NULL,
    monto_pagado DECIMAL(10,2) NOT NULL,
    fecha_pago DATE NOT NULL
);
-- CREACIÓN DE TABLA DE LOGS
CREATE TABLE logs (
    id INT PRIMARY KEY AUTO_INCREMENT,
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    tipo_evento VARCHAR(50) NOT NULL,
    data TEXT NOT NULL
);
-- PROCEDIMIENTO ALMACENADO PARA REGISTRAR UN NUEVO RECIBO DE PAGO
DELIMITER $$
CREATE PROCEDURE RegistrarRecibo (
    IN p_id_usuario INT,
    IN p_id_factura INT,
    IN p_monto_pagado DECIMAL(10,2),
    IN p_fecha_pago DATE
)
BEGIN
    -- Insertar el recibo de pago
    INSERT INTO Recibos_De_Pago (id_usuario, id_factura, monto_pagado, fecha_pago)
    VALUES (p_id_usuario, p_id_factura, p_monto_pagado, p_fecha_pago);
END$$
DELIMITER ;
-- TRIGGER PARA REGISTRAR LOG CUANDO SE INSERTA UN NUEVO RECIBO
DELIMITER $$
CREATE TRIGGER after_insert_recibo
AFTER INSERT ON Recibos_De_Pago
FOR EACH ROW
BEGIN
    INSERT INTO logs (tipo_evento, data)
    VALUES ('Inserción de Recibo', CONCAT('Pago registrado: ID ', NEW.id_pago, ', Usuario ', NEW.id_usuario, ', Monto ',
NEW.monto_pagado, ', Fecha ', NEW.fecha_pago));
END$$
DELIMITER ;
-- INSERCIÓN DE DATOS DE PRUEBA
INSERT INTO Recibos_De_Pago (id_usuario, id_factura, monto_pagado, fecha_pago) VALUES
(1, 101, 150.00, CURDATE()),
(2, 102, 200.00, CURDATE());

-- CONSULTA PARA VER LOS RESULTADOS DE LOS LOGS
SELECT * FROM logs;
```

Conclusiones

1. **El recibo está bien estructurado y fácil de entender**

Al revisar el recibo, muestra que la información está claramente organizada. Los datos significativos, como el consumo de agua, son que tienen que pagar y se incluyen fechas importantes. Esto ayuda a los usuarios fáciles de entender cuánto tienen que pagar y por qué.

2. **Se identifican los elementos de facturación más importantes**

El análisis del documento revelamos que la recopilación de la colección se basa en leer el medidor, ciertos precios y algunos costos adicionales. Es importante que estos datos sean precisos para evitar las tarifas o requisitos de los usuarios incorrectos.
Información sobre la digitalización de la gestión mejorada

3. **Actualmente, el recibo muestra muchos datos que puede facilitar la entrada al ingresar un sistema digital o una base de datos.**

Esto reduciría los errores, aceleraría los procesos de pago y garantizaría un mejor control de consumo.
Las condiciones entre los datos permiten una estructura efectiva

4. **Analizar la información de recibo**

Puede definir conexiones entre usuarios, consumo, tarifas y pagos. Ayuda a elaborar una base de datos organizada que le permita consultar rápidamente y tomar decisiones mejores.

5. **Tener la capacidad de mejorar los datos con el análisis de datos**

Si la información se almacena en un sistema bien estructurado, los informes de consumo pueden registrar patrones de uso e incluso ofrecer a los usuarios recomendaciones para optimizar su consumo de agua. Sería beneficioso tanto para la empresa como para los clientes.

Github Enlaces

https://github.com/kilvet/UMH-BD2-2025P1-202100176

https://github.com/dago20201/UMH-BD2-2025P1-202202541

https://github.com/Ales2704/-UMH-BD2-2025P1-202301761

https://github.com/dzroberto72/UMH-BD2-2025P1-202401986