Sistema de Gestión para Restaurante: Implementación y Evaluación

Cruz Miranda Luis Eduardo
De la Rosa Lara Gustavo
Dunzz Llampallas Alan
González Arredondo Rafael
Lemus Gonzales Javier Isaac
Moreno Santoyo Mariana

Facultad de Ingeniería
Universidad Nacional Autónoma de México
Ciudad de México, México

Equipo: Seguidores de Fernando Arreola

***Resumen*—Este documento presenta el desarrollo y la implementación de un sistema de gestión diseñado para mejorar las operaciones diarias de un restaurante. El sistema abarca funciones críticas como la gestión de inventarios, pedidos y recursos humanos.**

***Index Terms*—gestión de restaurantes, base de datos, sistema de información, PostgreSQL**

I. Introducción

Este proyecto final para el curso de Bases de Datos de la carrera de Ingeniería en Computación en la Universidad Nacional Autónoma de México busca aplicar de manera práctica los conocimientos adquiridos a lo largo del semestre en un contexto real y tangible. A través del diseño y desarrollo de un sistema de gestión para restaurantes, el proyecto no solo mejora la eficiencia operativa mediante la digitalización de procesos sino que también proporciona una plataforma robusta para la gestión de información crucial en la toma de decisiones.

En la industria restaurantera actual, caracterizada por una competencia feroz y margenes estrechos, la eficiencia operativa y la satisfacción del cliente son más críticas que nunca. Los avances tecnológicos han abierto nuevas posibilidades para la optimización de recursos y la personalización de la experiencia del cliente, haciendo de sistemas de información bien diseñados un componente esencial del éxito empresarial.

Desde la perspectiva académica, este proyecto permite una exploración profunda de las teorías de bases de datos y su aplicación práctica. Al enfrentar desafíos reales en el diseño del sistema, los estudiantes no solo aplican sus conocimientos teóricos sino que también desarrollan habilidades críticas en resolución de problemas y pensamiento crítico, esenciales para su futura carrera profesional.

El proyecto se inició con un análisis meticuloso de los requerimientos, seguido por el desarrollo y refinamiento de varios modelos Entidad-Relación. Este proceso culminó en un diseño que fue rigurosamente validado por el profesor, asegurando su alineación con las necesidades operativas del restaurante y los objetivos educativos del curso. Los desafíos técnicos encontrados durante la implementación, tales como la optimización de consultas y la seguridad de los datos, fueron abordados mediante soluciones innovadoras que demostraron la aplicación efectiva de principios avanzados de bases de datos.

Además, el impacto de este proyecto trasciende el aula; prepara a los estudiantes para desafíos profesionales y refuerza su capacidad para contribuir significativamente en sus futuros roles como ingenieros en computación. En última instancia, el proyecto no solo cumple con los requisitos operacionales del restaurante sino que también ilustra la importancia de las bases de datos en la estrategia empresarial y la operación diaria.

II. Análisis del Problema

Al comienzo del proyecto, el equipo realizó una lectura detallada y un análisis exhaustivo del documento de requerimientos proporcionado por el profesor. Este documento, estructurado en dos partes principales, fue esencial para identificar y entender los desafíos específicos que enfrenta la gestión de un restaurante y cómo un sistema de gestión de bases de datos podría abordar estos problemas de manera efectiva.

II-A. Parte Uno: Requerimientos Operativos y Funcionales

La primera parte del documento detallaba los requerimientos operativos y funcionales necesarios para el sistema. A través de este análisis, identificamos varios desafíos clave:

- **Gestión Compleja de Empleados:** Se necesitaba un sistema que maneje de forma eficiente y segura la información detallada de los empleados, incluyendo múltiples roles y especificaciones como horarios de meseros y especialidades de cocineros.
- **Control de Inventario y Pedidos:** El documento resaltó la necesidad de un control riguroso de inventario y un

sistema de pedidos que asegure la disponibilidad y la correcta facturación de platillos y bebidas.

II-B. Parte Dos: Implementación y Visualización de Datos

La segunda parte del documento se enfocaba en la implementación y cómo los usuarios finales interactuarían con el sistema. De esta sección, extrajimos la necesidad de desarrollar interfaces accesibles y funcionalidades específicas:

- **Interfaces de Usuario Intuitivas:** Era crucial diseñar interfaces que permitieran a los empleados gestionar y consultar información fácilmente, lo que incluía la visualización de datos en tiempo real a través de dashboards.
- **Funcionalidades de Reporte y Análisis:** Se identificó la importancia de incluir capacidades avanzadas de reporte y análisis para mejorar la toma de decisiones estratégicas basadas en datos recopilados y procesados por el sistema.

II-C. Conclusión del Análisis

Este análisis preliminar del documento proporcionado por el profesor fue fundamental para establecer una base sólida para el desarrollo del sistema. Nos permitió comprender completamente los requisitos y expectativas, asegurando que el diseño del sistema no solo cumpliera con los objetivos operativos y administrativos del restaurante sino que también se alineara con los objetivos educativos del curso. La próxima fase del proyecto involucraría el diseño detallado del sistema, tomando en cuenta todas estas necesidades y desafíos identificados.

III. Propuesta de Solución

El desarrollo de la base de datos para el sistema de gestión del restaurante implicó un proceso metodológico y estructurado, que comenzó con la creación de un modelo Entidad-Relación inicial y culminó con la validación de un modelo relacional completo por parte del profesor del curso.

III-A. Desarrollo del Modelo Entidad-Relación

El primer paso en la propuesta de solución fue la implementación de un modelo Entidad-Relación. Este modelo fue esencial para visualizar y estructurar las diversas entidades que componen el sistema del restaurante y sus interrelaciones. Durante esta fase, el equipo creó múltiples borradores del modelo, refinándolos progresivamente a través de discusiones grupales y sesiones de revisión. Cada iteración buscaba mejorar la representación de las relaciones complejas y las reglas de negocio inherentes al funcionamiento del restaurante.

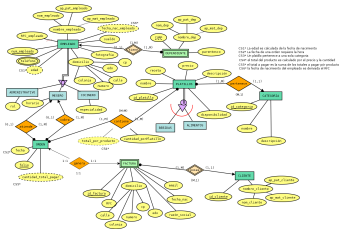
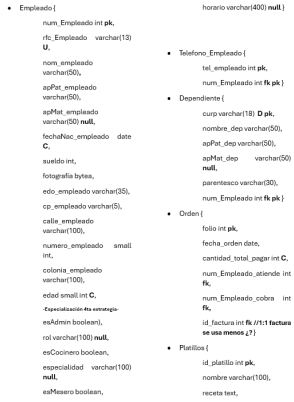


Figura 1. MODELO ENTIDAD RELACIÓN

III-B. Transición al Modelo Relacional Intermedio

Una vez que el modelo Entidad-Relación fue suficientemente robusto y detallado, el equipo procedió a convertirlo en un modelo relacional intermedio. Este paso fue crucial para transformar las abstracciones del modelo Entidad-Relación en un esquema que pudiera ser implementado en un sistema de gestión de bases de datos. El modelo relacional intermedio sirvió como un puente entre los conceptos teóricos y la aplicación práctica, ajustando las entidades para optimizar el almacenamiento de datos y la eficiencia de las consultas.



- Empleado {
  num_Empleado int **pk,**
  rfc_Empleado varchar(13) **U,**
  nom_empleado varchar(50),
  apPat_empleado varchar(50),
  apMat_empleado varchar(50) **null,**
  fechaNac_empleado date **C,**
  sueldo int,
  fotografia bytea,
  edo_empleado varchar(35),
  cp_empleado varchar(5),
  calle_empleado varchar(100),
  numero_empleado small int,
  colonia_empleado varchar(100),
  edad small int **C,**
  -Especialización-Es estrategia-
  esAdmin boolean),
  rol varchar(100) **null,**
  esCocinero boolean,
  especialidad varchar(100) **null,**
  esMesero boolean,
  horario varchar(400) **null** }
- Telefono_Empleado {
  tel_empleado int **pk,**
  num_Empleado int **fk pk** }
- Dependiente {
  curp varchar(18) **D pk,**
  nombre_dep varchar(50),
  apPat_dep varchar(50),
  apMat_dep varchar(50) **null,**
  parentesco varchar(30),
  num_Empleado int **fk pk** }
- Orden {
  folio int **pk,**
  fecha_orden date,
  cantidad_total_pagar int **C,**
  num_Empleado_atiende int **fk,**
  num_Empleado_cobra int **fk,**
  id_factura int **fk //1:1 factura se usa menos ¿?** }
- Platillos {
  id_platillo int **pk,**
  nombre varchar(100),
  receta text,

Figura 2. RELACIONAL INTERMEDIO

III-C. Finalización con el Modelo Relacional

El modelo relacional intermedio fue entonces refinado hasta llegar a un modelo relacional definitivo. Este modelo incorporó todas las características necesarias para una implementación efectiva, incluyendo la definición de claves primarias y foráneas, índices para mejorar el rendimiento y restricciones para garantizar la integridad de los datos. Cada aspecto del modelo relacional fue exhaustivamente revisado y ajustado para alinear con los requisitos operativos y de negocio del restaurante.

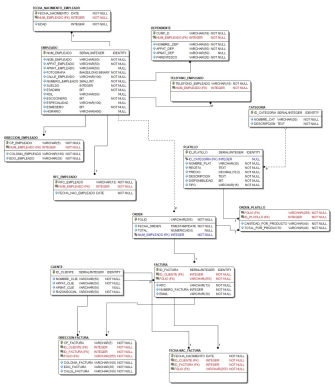


Figura 3. MODELO RELACIONAL

III-D. *Validación del Modelo por el Profesor*

El modelo relacional final fue presentado al profesor para su validación. Este paso fue decisivo para asegurar que el modelo no solo cumpliera con los estándares académicos del curso sino que también fuera viable para la implementación práctica. La aprobación del profesor confirmó que el modelo estaba bien diseñado y preparado para proceder con la fase de implementación en el entorno de base de datos utilizando PostgreSQL y pgAdmin.

III-E. *Conclusión*

Este enfoque estructurado y metódico aseguró que la base de datos desarrollada no solo sea funcional y eficiente sino que también robusta y escalable. La solución propuesta ahora sirve como el fundamento sobre el cual el sistema de gestión del restaurante operará, permitiendo mejoras continuas y adaptaciones a medida que las necesidades del restaurante evolucionen.

IV. Implementación

V. Implementación de la Base de Datos

La implementación de la base de datos para el sistema de gestión del restaurante se dividió en tres partes principales, con el objetivo de organizar y simplificar el proceso de desarrollo y mantenimiento. Este enfoque modular facilita la gestión de cambios y la revisión del sistema, permitiendo actualizaciones específicas sin afectar el funcionamiento global. A continuación, se describe detalladamente cada uno de los archivos utilizados para la implementación:

V-A. *Archivo de Creación de Tablas*

El archivo de creación de tablas constituye la base estructural de la base de datos. En este archivo, se definen todas las tablas necesarias para el sistema, junto con sus campos, tipos de datos, y las relaciones entre ellas, incluyendo claves primarias y foráneas, restricciones y otros atributos necesarios para garantizar la integridad y el rendimiento de la base de datos. Este archivo es crucial porque establece cómo se organizarán y almacenarán los datos dentro del sistema. La creación de tablas incluye:

- **Tabla Empleado:** Almacena información de los empleados del restaurante, incluyendo detalles personales y laborales.

```
CREATE TABLE Empleado (
    num_empleado SERIAL NOT NULL,  -- N mero de empleado como clave primaria
    nom_empleado VARCHAR(50) NOT NULL,  -- Nombre del empleado
    apPat_empleado VARCHAR(50) NOT NULL,  -- Apellido paterno del empleado
    apMat_empleado VARCHAR(50) NULL,  -- Apellido materno del empleado
    fotografia VARCHAR(255) NOT NULL,  -- Fotograf a del empleado en formato binario
    calle_empleado VARCHAR(100) NOT NULL,  -- Calle de la direcci n del empleado
    numero_empleado SMALLINT NOT NULL,  -- N mero de la direcci n del empleado
    sueldo INT NOT NULL,  -- Sueldo del empleado
    esAdmin BOOLEAN NOT NULL,  -- Indica si el empleado es administrador
    rol VARCHAR(50) NULL,  -- Rol del empleado
    esCocinero BOOLEAN NOT NULL,  -- Indica si el empleado es cocinero
    especialidad VARCHAR(100) NULL,  -- Especialidad del empleado (si es cocinero)
    esMesero BOOLEAN NOT NULL,  -- Indica si el empleado es mesero
    horario VARCHAR(400) NULL,  -- Horario del empleado
    CONSTRAINT num_empleado_PK PRIMARY KEY (num_empleado)  -- Definici n de la clave primaria
);


```

- **Tabla Cliente:** Registra los datos de los clientes para facilitar operaciones como la facturación y el seguimiento de preferencias.

```

        CREATE TABLE Cliente (
            id_cliente SERIAL NOT NULL,  -- ID del cliente como clave primaria
            nombre_clie VARCHAR(50) NOT NULL,  -- Nombre del cliente
            apPat_clie VARCHAR(50) NOT NULL,  -- Apellido paterno del cliente
            apMat_clie VARCHAR(50) NULL,  -- Apellido materno del cliente
            razonSocial VARCHAR(50) NOT NULL,  -- Raz n social del cliente
            CONSTRAINT cliente_FK PRIMARY KEY (id_cliente)  -- Definici n de la clave primaria
);

```

- **Tabla Platillo:** Contiene información sobre los platillos ofrecidos por el restaurante, esencial para la gestión del menú y el control de inventario.

```

CREATE TABLE Platillo (
    id_platillo SERIAL NOT NULL,  -- ID del platillo como clave primaria
    nombre_plat VARCHAR(30) NOT NULL,  -- Nombre del platillo
    receta TEXT NOT NULL,  -- Receta del platillo
    precio DECIMAL(10,2) NOT NULL,  -- Precio del platillo
    descripcion TEXT NOT NULL,  -- Descripci n del platillo
    disponibilidad BOOLEAN NOT NULL,  -- Disponibilidad del platillo
    tipo VARCHAR(15) NOT NULL,  -- Tipo de platillo
    id_categoria INT NOT NULL,  -- ID de la categor a a la que pertenece el platillo
    CONSTRAINT platillo_PK PRIMARY KEY (id_platillo),  -- Definici n de la clave primaria
    CONSTRAINT platillo_categoria_FK FOREIGN KEY (id_categoria)  -- Definici n de la clave for nea
        REFERENCES Categoria (id_categoria) ON DELETE SET NULL ON UPDATE CASCADE
);

```

- **Tabla Orden:** Captura detalles de las órdenes realizadas por los clientes, vinculando platillos, precios y empleados que gestionan la orden.

```
CREATE TABLE Orden (
    folio VARCHAR(255) NOT NULL CHECK (folio
    LIKE 'ORD-%'),  -- Folio de la orden con
    restricci n de formato
    fecha_orden TIMESTAMP NOT NULL,  -- Fecha de
     la orden
    total NUMERIC NULL,  -- Total de la orden
    num_empleado INT NOT NULL,  -- N mero de
    empleado que realiz  la orden
    CONSTRAINT folio_PK PRIMARY KEY (folio),  --
     Definici n de la clave primaria
    CONSTRAINT orden_empleado_FK FOREIGN KEY (
    num_empleado)  -- Definici n de la clave
    for nea
        REFERENCES Empleado (num_empleado) ON
    DELETE CASCADE ON UPDATE RESTRICT
);

```

- **Tabla Factura:**La tabla Factura gestiona los registros de todas las transacciones financieras relacionadas con las ventas a los clientes. Almacena información detallada de cada factura, incluyendo el enlace a la orden correspondiente y los detalles necesarios para cumplir con las obligaciones fiscales.

```
CREATE TABLE Factura (
    id_factura SERIAL PRIMARY KEY,  --
    Identificador  nico  para cada factura.
    id_cliente INT NOT NULL,  -- ID del cliente
    que relaciona la factura con datos del
    cliente.
    folioOrden VARCHAR(255),  -- Folio de la
    orden asociada a la factura para
    correlaci n.
    rfc VARCHAR(13) NOT NULL,  -- RFC del
    cliente para fines de facturaci n fiscal.
    numero_factura INT NOT NULL,  -- N mero
    secuencial de la factura,  nico  para cada
    factura.
    email VARCHAR(30) NOT NULL,  -- Email del
    cliente para enviar detalles de la factura
    electr nicamente.
    CONSTRAINT factura_cliente_FK FOREIGN KEY (
    id_cliente)
        REFERENCES Cliente (id_cliente) ON
    DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT unique_id_factura UNIQUE (
    id_factura),  -- Asegura que cada factura
    tenga un identificador  nico .
    CONSTRAINT factura_orden_PK FOREIGN KEY (
    folioOrden)
        REFERENCES Orden(folio) ON DELETE
    CASCADE ON UPDATE CASCADE  -- Enlace con la
     tabla Orden para integridad de datos.
);
```

Las tablas descritas anteriormente representan solo una selección de las estructuras más fundamentales y cruciales dentro de la base de datos del sistema de gestión del restaurante. Estas tablas han sido destacadas por su papel central en la operación diaria y la gestión administrativa del restaurante. A continuación, profundizaremos en la implementación y el uso de triggers, los cuales son esenciales para mantener la integridad de los datos y automatizar procesos clave dentro de la base de datos.

VI. Uso de Triggers en la Base de Datos

Los triggers son componentes fundamentales en la administración de bases de datos, diseñados para automatizar tareas y asegurar la integridad y la consistencia de los datos. En el sistema de gestión del restaurante, utilizamos diversos triggers para manejar operaciones automáticas que son cruciales para la precisión de la información financiera y operacional.

VI-A. Trigger para Actualizar el Total por Producto

Descripción del Trigger: Este trigger, 'actualizar_totalPorProducto', está diseñado para calcular de forma automática el total de cada producto en la tabla 'Orden_Platillo' cada vez que se inserta o actualiza un registro en dicha tabla. Su función es asegurar que el total refleje correctamente el precio actual del platillo multiplicado por la cantidad pedida.

Código SQL y Comentarios Detallados:

```
-- Definici n de la funci n de trigger
CREATE OR REPLACE FUNCTION
    actualizar_totalPorProducto() RETURNS TRIGGER AS
     $$
BEGIN
    -- Calcular el total por producto multiplicando
    la cantidad del producto por su precio actual
    NEW.totalPorProducto := NEW.cantidadPorProducto
    * (SELECT precio FROM Platillo WHERE id_platillo
     = NEW.id_platillo);
    -- Devolver el registro nuevo con el total
    actualizado para aplicar los cambios
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Creaci n del trigger que invoca la funci n
    antes mencionada
CREATE TRIGGER trigger_actualizar_totalPorProducto
BEFORE INSERT OR UPDATE ON Orden_Platillo
FOR EACH ROW
EXECUTE FUNCTION actualizar_totalPorProducto();
```

VI-B. Trigger para Actualizar el Total de la Orden

Descripción del Trigger: El trigger 'actualizar_totalOrden' es responsable de actualizar el total general de cada orden en la tabla 'Orden' cada vez que se inserta o actualiza un platillo en dicha orden. Este proceso garantiza que el total en la orden sea siempre la suma correcta de los productos incluidos. **Código SQL y Comentarios Detallados:**

```
-- Definici n de la funci n de trigger
CREATE OR REPLACE FUNCTION actualizar_totalOrden()
    RETURNS TRIGGER AS $$
BEGIN
    -- Recalcular el total general de la orden
    sumando todos los totales por producto de esa
    orden
    NEW.total := (SELECT SUM(totalPorProducto) FROM
    Orden_Platillo WHERE folio = NEW.folio);
    -- Devolver el registro nuevo con el total
    actualizado para aplicar los cambios
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Creaci n del trigger que invoca la funci n
    antes mencionada
```

```
CREATE TRIGGER trigger_actualizar_totalOrden
BEFORE INSERT OR UPDATE ON Orden
FOR EACH ROW
EXECUTE FUNCTION actualizar_totalOrden();
```

Estas secciones detalladas proporcionan un entendimiento claro de la funcionalidad específica de cada trigger, destacando cómo contribuyen a la automatización y la eficiencia del sistema de gestión del restaurante. Esta organización ayuda a los lectores a entender mejor el propósito y el funcionamiento de cada parte de la base de datos relacionada con los triggers.

VII. Implementación de Índices

La implementación de índices en la base de datos es crucial para mejorar el rendimiento de las consultas y era un requisito específico para asegurar una recuperación de datos eficiente y rápida. Los índices permiten que el sistema de gestión de bases de datos localice rápidamente los datos sin tener que buscar en cada fila de una tabla cada vez que se realiza una consulta.

VII-A. Índice en la Columna de Empleados

Descripción del Índice: El índice 'i_num_empleado' en la tabla 'Empleado' mejora significativamente la velocidad de las operaciones de búsqueda y consulta que involucran el número de empleado, que es una operación común en muchas funciones administrativas y operacionales del sistema.

Código SQL y Comentarios Detallados:

```
-- Creaci n de un  ndice  hash en la columna
    num_empleado de la tabla Empleado
-- Los  ndices  hash son particularmente eficientes
    para operaciones de b squeda que utilizan la
    igualdad
CREATE INDEX i_num_empleado ON Empleado USING hash (
    num_empleado);
```

Justificación del Uso del Índice: Este índice fue implementado como parte de los requisitos para optimizar las consultas que involucran identificar empleados específicos por su número de identificación. Al utilizar un índice hash, el sistema puede acceder directamente a los datos del empleado sin necesidad de un escaneo completo de la tabla, lo que reduce el tiempo de consulta y aumenta la eficiencia del sistema.

La utilización de este índice es particularmente relevante para funciones como el procesamiento de la nómina, la administración de recursos humanos y las consultas de seguridad del empleado, donde la rapidez y la precisión son cruciales.

VII-B. Archivo de Inserción de Datos

VIII. Archivo de Inserción de Datos

El archivo de inserción de datos juega un papel crucial en la población inicial del sistema de base de datos del restaurante con información relevante que permite su operación diaria. Este archivo contiene funciones y comandos SQL que insertan registros en diversas tablas, asegurando que la base de datos esté lista para su uso tras la configuración inicial.

VIII-A. Función para Agregar Empleados

Descripción de la Función: La función 'agregar_empleado' es una parte esencial del archivo de inserción de datos. Esta función permite la inserción estructurada y eficiente de nuevos empleados en la base de datos, cubriendo todos los aspectos relevantes desde la información personal y laboral hasta detalles específicos como el RFC y la dirección.

Código SQL y Comentarios Detallados:

```
-- Definici n de la funci n para agregar un nuevo
    empleado al sistema
CREATE OR REPLACE FUNCTION agregar_empleado(
    IN p_nom_empleado VARCHAR(50),
    IN p_apPat_empleado VARCHAR(50),
    IN p_apMat_empleado VARCHAR(50),
    IN p_fotografia VARCHAR(255),
    IN p_calle_empleado VARCHAR(100),
    IN p_numero_empleado SMALLINT,
    IN p_sueldo INT,
    IN p_esAdmin BOOLEAN,
    IN p_rol VARCHAR(50),
    IN p_esCocinero BOOLEAN,
  IN p_especialidad VARCHAR(100),
  IN p_esMesero BOOLEAN,
  IN p_horario VARCHAR(400),
  IN p_rfc_empleado VARCHAR(13),
  IN p_fecha_nacimiento DATE,
  IN p_cp_empleado VARCHAR(5),
  IN p_colonia_empleado VARCHAR(100),
  IN p_edo_empleado VARCHAR(35),
  IN p_edad INT,
  IN p_telefono_empleado VARCHAR(15),
    OUT p_num_empleado INT
)
AS $$
BEGIN
    -- Insertar informaci n b sica del empleado en
     la tabla Empleado
    INSERT INTO Empleado (nom_empleado,
    apPat_empleado, apMat_empleado, fotografia,
    calle_empleado,
  numero_empleado, sueldo, esAdmin, rol, esCocinero,
     especialidad, esMesero, horario)
    VALUES (p_nom_empleado, p_apPat_empleado,
    p_apMat_empleado, p_fotografia, p_calle_empleado
    ,
  p_numero_empleado, p_sueldo, p_esAdmin, p_rol,
    p_esCocinero, p_especialidad, p_esMesero,
    p_horario)
    RETURNING num_empleado INTO p_num_empleado;

    -- Insertar el RFC y la fecha de nacimiento del
    empleado
    INSERT INTO rfc_empleado (rfc_empleado,
    fechaNac_empleado, num_empleado)
    VALUES (p_rfc_empleado, p_fecha_nacimiento,
    p_num_empleado);

    -- Registrar la direcci n del empleado
    INSERT INTO direccion_empleado (cp_empleado,
    colonia_empleado, edo_empleado, num_empleado)
    VALUES (p_cp_empleado, p_colonia_empleado,
    p_edo_empleado, p_num_empleado);

    -- Insertar datos de nacimiento y edad
    INSERT INTO fecha_nacimiento_empleado (
    fecha_nacimiento, edad, num_empleado)
    VALUES (p_fecha_nacimiento, p_edad,
    p_num_empleado);

    -- Registrar el tel fono del empleado
```

```
    INSERT INTO Telefono_Empleado (num_empleado,
    telefono_empleado)
    VALUES (p_num_empleado, p_telefono_empleado);

END;
$$ LANGUAGE plpgsql;
```

Este procedimiento automatizado asegura que todos los datos necesarios se inserten de manera coherente y sin errores, facilitando la gestión de recursos humanos y el acceso a información completa del personal cuando sea necesario.

VIII-B. Ejemplo de Uso de la Función para Agregar Empleados

Para ilustrar cómo se puede utilizar la función 'agregar_empleado' para insertar nuevos empleados en la base de datos, presentamos el siguiente ejemplo práctico. Este ejemplo demuestra la inserción de un nuevo empleado, un cocinero especializado en mariscos, utilizando la función definida previamente.

Código SQL y Explicación de Parámetros:

```
-- Llamada a la funci n agregar_empleado con
    valores espec ficos para cada par metro
SELECT agregar_empleado(
    'Axel'::varchar,                          -- Nombre
     del empleado
    'Aguilar'::varchar,                       --
    Apellido paterno del empleado
    'Goicochea'::varchar,                     --
    Apellido materno del empleado
    'https://thispersondoesnotexist.com'::varchar,
    -- URL de una imagen de ejemplo para la
    fotograf a
    'Cerro de Chapultepec'::varchar,          -- Calle
    de la direcci n del empleado
    304::smallint,                            --
    N mero de casa o edificio
    10000,                                    -- Sueldo
     asignado al empleado
    FALSE,                                    -- Indica
     que el empleado no tiene privilegios de
    administrador
    NULL::varchar,                            -- Rol no
     especificado (NULL)
    TRUE,                                     -- Indica
     que el empleado es cocinero
    'Mariscos'::varchar,                      --
    Especialidad culinaria del empleado
    FALSE,                                    -- Indica
     que el empleado no es mesero
    NULL::varchar,                            --
    Horario no especificado (NULL)
    'AUGA010903D6A'::varchar,                 -- RFC
    del empleado
    '2001-09-03'::date,                       -- Fecha
    de nacimiento del empleado
    '14370'::varchar,                         --
    C digo postal de la direcci n del empleado
    'Residencial Chimali'::varchar,           --
    Colonia de la direcci n del empleado
    'Ciudad de M xico'::varchar,               --
    Estado de la direcci n del empleado
    23,                                       -- Edad
    del empleado
    '5512452809'::varchar                     --
    Tel fono del empleado
);
```

Este ejemplo demuestra cómo se pueden llenar todos los campos necesarios para registrar a un empleado en el sistema utilizando una función almacenada. Esto simplifica el proceso de inserción y asegura que todos los datos necesarios sean incluidos y validados correctamente, reduciendo errores y manteniendo la integridad de la base de datos.

VIII-C. Ejemplo de Inserción de Datos en la Tabla Cliente

A continuación, se muestra un ejemplo práctico de cómo insertar un nuevo cliente en la base de datos. Este ejemplo es fundamental para entender cómo se pueden añadir registros de forma manual a la tabla 'Cliente' utilizando la instrucción SQL 'INSERT INTO'.

Código SQL:

```
INSERT INTO Cliente (
    nombre_clie,      -- Columna para el nombre del
    cliente
    apPat_clie,       -- Columna para el apellido
    paterno del cliente
    apMat_clie,       -- Columna para el apellido
    materno del cliente
    razonSocial       -- Columna para la raz n social
     del cliente
) VALUES(
    'Alonso',         -- Valor para el nombre del
    cliente
    'Hern ndez',      -- Valor para el apellido
    paterno
    'Morales',        -- Valor para el apellido
    materno
    'Hexagon'         -- Valor para la raz n social
);
```

Este comando SQL inserta un nuevo registro en la tabla 'Cliente', proporcionando valores específicos para el nombre, los apellidos y la razón social del cliente. Es un procedimiento estándar para añadir datos y es crucial para el mantenimiento de la base de datos actualizada y precisa.

VIII-D. Ejemplo de Inserción de Datos en la Tabla Platillo

Este ejemplo muestra cómo añadir un nuevo platillo a la base de datos en la tabla 'Platillo'. Se detalla cómo proporcionar cada pieza de información requerida para el registro de un platillo, desde su nombre hasta su categorización.

Código SQL:

```
INSERT INTO Platillo (
    nombre_plat,      -- Columna para el nombre del
    platillo
    receta,           -- Columna para la receta del
    platillo
    precio,           -- Columna para el precio del
    platillo
    descripcion,      -- Columna para la descripci n
    del platillo
    disponibilidad,   -- Columna para indicar si el
    platillo est  disponible
    tipo,             -- Columna para el tipo de
    platillo
    id_categoria      -- Clave for nea que vincula el
     platillo con su categor a
) VALUES(
    'Molletes',       -- Nombre del platillo
    '1 bolillo con frijoles, queso manchego, cebolla
    , jitomate y chile picado', -- Receta detallada
    del platillo
    40.0,             -- Precio del platillo
    '1 bolillo con frijoles, queso manchego, cebolla
    , jitomate y chile picado', -- Descripci n del
    platillo
```

```
    TRUE,              -- El platillo est  disponible
    'Platillo',        -- Tipo de platillo
    1                  -- ID de la categor a a la que
    pertenece el platillo
);
```

Este comando SQL inserta un nuevo platillo llamado "Molletes.[en] la base de datos. Se proporciona una descripción completa del platillo que incluye su composición y preparación, el precio, su disponibilidad y la categoría a la que pertenece. Este ejemplo ilustra la manera de añadir registros a la tabla 'Platillo' que es esencial para el manejo del menú en el restaurante.

IX. Descripción de la Función 'agregar_orden'

La función 'agregar_orden' implementada en PostgreSQL desempeña un papel crucial en la automatización del proceso de gestión de órdenes en el sistema de un restaurante. Esta función está diseñada para facilitar la inserción eficiente de múltiples platillos dentro de una única orden, integrando diversos parámetros que capturan la esencia de cada transacción realizada.

IX-A. Parámetros de la Función

La función acepta los siguientes parámetros de entrada, que son esenciales para especificar los detalles de cada orden:

- **p_folio**: Identificador único para cada orden, usualmente un string que puede incluir números y letras.
- **p_num_empleado**: Número de identificación del empleado que está registrando la orden.
- **p_id_platillos**: Array de enteros que contiene los IDs de los platillos que se están ordenando.
- **p_cantidadesPorProducto**: Array de enteros pequeños (SMALLINT) que contiene las cantidades correspondientes para cada platillo indicado en *p_id_platillos*.

IX-B. Proceso y Funcionalidad de la Función

La función procede de la siguiente manera para gestionar la inserción de una orden:

1. **Inserción en la tabla 'Orden'**: Inicialmente, la función inserta un registro en la tabla 'Orden', asignando el folio proporcionado, la fecha y hora actuales del sistema, y el número de empleado. Este paso establece la base para los detalles subsiguientes de la orden.

```
    INSERT INTO Orden (folio, fecha_orden,
    num_empleado)
    VALUES (p_folio, CURRENT_TIMESTAMP,
    p_num_empleado);

```

2. **Iteración sobre los platillos**: Utilizando un bucle controlado por el tamaño del array *p_id_platillos*, la función verifica la disponibilidad de cada platillo antes de proceder con su inserción. Si el platillo está disponible, se inserta en la tabla 'Orden_Platillo'; si no, se emite una notificación.

```
    FOR i IN 1..array_length(p_id_platillos, 1)
     LOOP
        IF ((SELECT disponibilidad FROM
    platillo WHERE id_platillo = p_id_platillos
    [i])) THEN
            INSERT INTO Orden_Platillo (folio,
    id_platillo, cantidadPorProducto)
            VALUES (p_folio, p_id_platillos[i],
     p_cantidadesPorProducto[i]);
        ELSE
            RAISE NOTICE 'Producto % no
    disponible', p_id_platillos[i];
        END IF;
    END LOOP;

```

3. **Actualización del total de la orden**: Finalmente, la función actualiza el total de la orden a 0 como valor inicial, permitiendo que un trigger posterior calcule el total real basado en los platillos incluidos y sus precios.

```
    UPDATE Orden SET total = 0 WHERE folio =
    p_folio;

```

Esta función encapsula la lógica necesaria para la inserción compleja de órdenes y demuestra el uso eficiente de los triggers y procedimientos almacenados dentro de PostgreSQL para gestionar las transacciones en un entorno de restaurante.

IX-C. Ejemplo de Uso de la Función 'agregar_orden'

Este ejemplo demuestra cómo se puede utilizar la función 'agregar_orden' para añadir una nueva orden al sistema de gestión del restaurante. A continuación, se presenta la llamada a la función con valores específicos para insertar una orden que incluye un platillo.

Código SQL y Explicación de la Llamada:

```
-- Llamada a la funci n agregar_orden con
    par metros espec ficos
SELECT agregar_orden(
    'ORD-001'::varchar,  -- folio de la orden,
    utilizado para identificarla de manera  nica
    1,                   -- num_empleado, el ID del
    empleado que registra la orden
    ARRAY[1],            -- id_platillo, un array
    que contiene el ID del platillo pedido
    ARRAY[2::smallint]   -- cantidadPorProducto, un
    array que indica la cantidad de cada platillo
    ordenado
);
```

Descripción de los Parámetros:

- **folio**: 'ORD-001', que es el identificador único para esta orden.
- **num_empleado**: 1, que corresponde al empleado que está registrando la orden.
- **id_platillo**: ARRAY[1], indica que el platillo con ID 1 está siendo ordenado.
- **cantidadPorProducto**: ARRAY[2::smallint], especifica que se están ordenando dos unidades del platillo con ID 1.

Este ejemplo muestra cómo la función facilita la inserción de detalles de la orden directamente desde el sistema frontend o de gestión, utilizando arrays para manejar múltiples platillos y sus cantidades en una sola transacción. Esto asegura que

el proceso de toma de órdenes sea eficiente y minimice los errores de entrada de datos, permitiendo un registro preciso y rápido de las transacciones en la base de datos.

X. Descripción de la Función 'agregar_factura'

La función 'agregar_factura' está diseñada para facilitar la inserción de registros relacionados con facturas en el sistema de base de datos, abarcando múltiples tablas que incluyen datos de facturación y dirección del cliente. Esta función centraliza el proceso de creación de facturas, garantizando que toda la información relevante se maneje de manera consistente y eficiente.

X-A. *Parámetros de la Función*

La función acepta diversos parámetros de entrada que especifican la información necesaria para la creación de una factura:

- **p_id_cliente**: Identificador del cliente al que se emite la factura.
- **p_numero_factura**: Número secuencial de la factura.
- **p_email**: Correo electrónico del cliente para envío de factura electrónica.
- **p_folioOrden**: Folio de la orden asociada a la factura.
- **p_rfc**: RFC del cliente, necesario para cumplimiento fiscal.
- **p_cp_factura**: Código postal de la dirección de facturación.
- **p_colonia_factura**: Colonia de la dirección de facturación.
- **p_edo_factura**: Estado de la dirección de facturación.
- **p_calle_factura**: Calle de la dirección de facturación.
- **p_fecha_nacimiento**: Fecha de nacimiento del cliente, usada para verificación de identidad.

X-B. *Funcionalidad de la Función*

La función procede de la siguiente manera para gestionar la inserción de una factura:

1. **Inserción en la tabla 'Factura'**: Registra los detalles básicos de la factura, incluyendo el cliente, número de factura, correo electrónico, folio de la orden, y RFC.

```
INSERT INTO Factura (id_cliente,
numero_factura, email, folioorden, rfc)
VALUES (p_id_cliente, p_numero_factura,
p_email, p_folioOrden, p_rfc)
RETURNING id_factura INTO p_id_factura;
```

2. **Inserción en la tabla 'Direccion_Factura'**: Añade la dirección específica donde se debe enviar la factura física o legal.

```
INSERT INTO Direccion_Factura (cp_factura,
colonia_factura, edo_factura, calle_factura
, id_factura, id_cliente)
VALUES (p_cp_factura, p_colonia_factura,
p_edo_factura, p_calle_factura,
p_id_factura, p_id_cliente);
```

3. **Inserción en la tabla 'Fecha_Nac_Factura'**: Asocia la fecha de nacimiento del cliente con la factura, importante para ciertos requerimientos legales o de verificación.

```
INSERT INTO Fecha_Nac_Factura (
fecha_nacimiento, id_factura, id_cliente)
VALUES (p_fecha_nacimiento, p_id_factura,
p_id_cliente);
```

Esta función simplifica el proceso de inserción de facturas, asegurando que todos los elementos relacionados estén correctamente vinculados y almacenados en la base de datos.

X-C. *Ejemplo de Uso de la Función 'agregar_factura'*

Este ejemplo muestra cómo se puede utilizar la función 'agregar_factura' para añadir una nueva factura al sistema de gestión del restaurante. Se presentan los detalles específicos para cada parámetro utilizado en la llamada de la función, demostrando cómo se inserta un registro de factura vinculado a una orden y a datos del cliente.

Código SQL y Explicación de la Llamada:

```
-- Llamada a la funci n agregar_factura con
    par metros espec ficos
SELECT agregar_factura(
    5,                  -- id_cliente: ID del cliente
     al que se emite la factura.
    1005,               -- numero_factura: N mero
    secuencial de la factura.
    'cliente5@email.com', -- email: Correo
    electr nico del cliente para env o de la
    factura electr nica.
    'ORD-005',          -- folioOrden: Folio de la
    orden asociada a la factura.
    '5678901234E',      -- rfc: RFC del cliente para
    cumplimiento fiscal.
    '56789',            -- cp_factura: C digo postal
     de la direcci n de facturaci n.
    'Oeste',            -- colonia_factura: Colonia
    de la direcci n de facturaci n.
    'Estado',           -- edo_factura: Estado de la
    direcci n de facturaci n.
    'Avenida Quinta',   -- calle_factura: Calle de la
     direcci n de facturaci n.
    '1975-01-30'::date -- fecha_nacimiento: Fecha de
     nacimiento del cliente, usada para
    verificaci n de identidad.
);
```

Este ejemplo ilustra cómo la función 'agregar_factura' facilita la inserción de detalles completos de la factura directamente desde el sistema de gestión, utilizando los parámetros para manejar múltiples aspectos de la información de facturación en una sola transacción. Esto asegura que el proceso de facturación sea eficiente, minimizando los errores de entrada de datos y permitiendo un registro preciso y rápido de las transacciones en la base de datos.

X-D. *Archivo de Consultas*

(Posteriormente, se describiría el archivo de consultas, detallando cómo se realizan las consultas para extraer, actualizar o eliminar información de la base de datos, según las necesidades operativas y de gestión del restaurante.)

Una vez definidas las tablas y realizados inserts en ellas como información muestra se realizaron las siguientes consultas como parte de los requerimientos de la base de datos.

- **Consulta ”datos-ordenes-mesero”:**

```
    CREATE OR REPLACE FUNCTION
     datos_ordenes_mesero(numEmpleado INT)
    RETURNS TABLE (
    nom_empleado VARCHAR(50),
    apPat_empleado VARCHAR(50),
    total_vendido NUMERIC,
    num_de_ordenes BIGINT,
    mensaje_error TEXT -- Mantener el tipo de
     dato como TEXT
) AS $$


```

Esta función toma un parámetro numEmpleado (un número entero) y devuelve una tabla con las columnas mostradas en el código anterior. La función utiliza una consulta común de expresión de tabla (CTE) llamada “es-mesero” para verificar si el empleado es un mesero. Luego, otra CTE llamada “ordenes-mesero” calcula el total vendido y el número de órdenes para el mesero especificado en el día actual. Finalmente, se seleccionan los datos relevantes y se muestra el mensaje de error según la condición. La función “datos-ordenes-mesero” obtiene los datos de ventas de un mesero específico en el día actual y verifica si el empleado es realmente un mesero.

- **Vista ”platilloMasVendido”:**

```
CREATE VIEW platilloMasVendido AS

WITH vecesVendido AS (
    SELECT id_platillo, SUM(cantidadporproducto)
     AS veces
    FROM orden_platillo
    GROUP BY id_platillo
)
SELECT
    nombre_plat,
    veces,
    receta,
    precio,
    platillo.descripcion,
    disponibilidad,
    CASE
        WHEN tipo = '1' THEN 'bebida'
        WHEN tipo = '2' THEN 'comida'
        ELSE 'otro' -- En caso de que haya otros
     tipos no especificados
    END AS tipo,
    nombre_cat
FROM vecesVendido
JOIN platillo ON vecesVendido.id_platillo =
    platillo.id_platillo
JOIN categoria ON platillo.id_categoria =
    categoria.id_categoria
WHERE veces = (SELECT MAX(veces) FROM
    vecesVendido); -- si hay empate imprime
    todos


```

Esta vista muestra información sobre el platillo más vendido. Utiliza una CTE llamada “vecesVendido” para calcular la cantidad de veces que se ha vendido cada platillo. Luego, se seleccionan los datos del platillo con la mayor cantidad de ventas. De manera general, la vista “platilloMasVendido” calcula y muestra el platillo que ha sido vendido la mayor cantidad de veces.

- **Vista ”productosNoDisponibles”:**

```
CREATE VIEW productosNoDisponibles AS (
SELECT * FROM Platillo WHERE (disponibilidad=
    FALSE);


```

Esta vista se crea para mostrar todos los platillos que no están disponibles en el menú.

- **Consulta: .ºbtener-ventas”:**

```
CREATE OR REPLACE FUNCTION obtener_ventas(
    fecha_inicio DATE, fecha_fin DATE DEFAULT
    NULL)
RETURNS TABLE (
    fecha DATE,
    numero_ventas INTEGER,
    total_ventas NUMERIC
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        DATE_TRUNC('day', fecha_orden)::date AS
    fecha,
        COUNT(*)::integer AS numero_ventas,
        SUM(total) AS total_ventas
    FROM Orden
    WHERE (DATE_TRUNC('day', fecha_orden)::date
    > fecha_inicio and fecha_fin is NULL)
       OR (fecha_fin IS NOT NULL AND DATE_TRUNC(
    'day', fecha_orden)::date BETWEEN
    fecha_inicio AND fecha_fin)
    GROUP BY DATE_TRUNC('day', fecha_orden)::
    date;
END;
$$ LANGUAGE plpgsql;


```

Esta función tiene como objetivo obtener el número total de ventas y el monto total de ventas en una fecha o en un intervalo de fechas.

- **Vista ”print-Factura”:**

```
CREATE VIEW print_Factura AS
SELECT
  'Factura' as Titulo,
    f.folioorden,
    o.fecha_orden,
    o.total,
    (c.nombre_clie || ' ' || c.appat_clie || ' '
     || COALESCE(c.apmat_clie, '')) AS
    nombreCompleto,
    f.rfc,
    c.razonsocial,
    nf.fecha_nacimiento,
    f.email,
    f.numero_factura AS telefono,
    (df.calle_factura || ', ' || df.
    colonia_factura || ', ' || df.edo_factura
    || ', ' || df.cp_factura) AS domicilio
FROM
    factura f
    NATURAL JOIN fecha_nac_factura nf
    NATURAL JOIN direccion_factura df
    NATURAL JOIN cliente c
    JOIN orden o ON f.folioorden = o.folio
```

```
WHERE
    f.id_factura = 2;
```

Esta vista se crea para mostrar la información de una factura específica, formateada de manera detallada.

XI. Vista grafica PHP

XII. Implementación de la Gestión de Empleados en PHP

En esta sección, se presenta la implementación de un sistema en PHP que permite gestionar y visualizar los datos de empleados de un restaurante. El sistema se conecta a una base de datos PostgreSQL para obtener y mostrar los datos. La implementación consta de tres partes principales: la página de inicio, la página que muestra la tabla de empleados con capacidad de filtrado, y el archivo CSS para el estilo de la página.

XII-A. Archivo `index.php`

El archivo `index.php` contiene la página de inicio que proporciona una interfaz simple para navegar a la página que muestra la tabla de empleados.

```
// index.php
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>P gina de Inicio</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Restaurante</h1>
        <p>Esta es la p gina de inicio. Use el bot n de a continuaci n para navegar.</p>
        <a href="basedatos.php"><button>Ver Tabla de empleados</button></a>
    </div>
</body>
</html>
```

XII-B. Archivo `basedatos.php`

El archivo `basedatos.php` contiene el código HTML y PHP necesario para mostrar la lista de empleados y permitir el filtrado por número de empleado.

```
// basedatos.php
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista de Empleados</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Lista de Empleados</h1>

        <!-- Formulario para filtrar por n mero de empleado -->
        <form method="GET" action="basedatos.php">
            <label for="num_empleado">Filtrar por N mero de Empleado:</label>
            <input type="text" id="num_empleado" name="num_empleado" placeholder="Ingrese n mero de empleado">
            <button type="submit">Filtrar</button>
        </form>

        <?php
        // Conexi n a la base de datos PostgreSQL
        $conexion = pg_connect("host=localhost dbname=proyecto user=postgres password=123456");

        if (!$conexion) {
            die("Error al conectar con la base de datos: " . pg_last_error());
        }

        // Obtener el n mero de empleado del formulario, si existe
        $num_empleado = isset($_GET['num_empleado']) ? $_GET['num_empleado'] : '';

        // Consulta SQL para obtener los datos de los empleados
        $query = "SELECT num_empleado, nom_empleado, appat_empleado, apmat_empleado, fotografia, calle_empleado, numero_empleado, sueldo, esadmin, rol, escocinero, especialidad, esmesero, horario FROM empleado";

        if ($num_empleado) {
            $query .= " WHERE num_empleado = '" . pg_escape_string($num_empleado) . "'";
        }

        // Ejecutar la consulta
        $consulta = pg_query($conexion, $query);

        if (!$consulta) {
            die("Error en la consulta: " . pg_last_error());
        }

        // Mostrar los resultados en una tabla HTML
        echo "<table>";
        echo "<tr><th>N mero de Empleado</th><th>Nombre</th><th>Apellido Paterno</th><th>Apellido Materno</th><th>Imagen</th><th>Calle</th><th>N mero</th><th>Sueldo</th><th>Es Administrador</th><th>Rol</th><th>Es Cocinero</th><th>Especialidad</th><th>Es Mesero</th><th>Horario</th></tr>";
        while ($row = pg_fetch_assoc($consulta)) {
            echo "<tr>";
            echo "<td>" . htmlspecialchars($row['num_empleado']) . "</td>";
            echo "<td>" . htmlspecialchars($row['nom_empleado']) . "</td>";
            echo "<td>" . htmlspecialchars($row['appat_empleado']) . "</td>";
            echo "<td>" . htmlspecialchars($row['apmat_empleado']) . "</td>";
            echo "<td><img src='" . htmlspecialchars($row['fotografia']) . "' alt='Imagen del Empleado'></td>";
            echo "<td>" . htmlspecialchars($row['calle_empleado']) . "</td>";
            echo "<td>" . htmlspecialchars($row['numero_empleado']) . "</td>";
            echo "<td>" . htmlspecialchars($row['sueldo']) . "</td>";
```

```
            echo "<td>" . htmlspecialchars($row['esadmin']) . "</td>";
            echo "<td>" . htmlspecialchars($row['rol']) . "</td>";
            echo "<td>" . htmlspecialchars($row['escocinero']) . "</td>";
            echo "<td>" . htmlspecialchars($row['especialidad']) . "</td>";
            echo "<td>" . htmlspecialchars($row['esmesero']) . "</td>";
            echo "<td>" . htmlspecialchars($row['horario']) . "</td>";
            echo "</tr>";
        }
        echo "</table>";

        // Liberar el resultado de la consulta y cerrar la conexi n
        pg_free_result($consulta);
        pg_close($conexion);
        ?>
        <a href="index.php"><button>Volver a la P gina de Inicio</button></a>
    </div>
</body>
</html>
```

XII-C. *Archivo* `styles.css`

El archivo `styles.css` contiene el estilo CSS utilizado para diseñar la apariencia de la página web.

```
// styles.css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
}

.container {
    width: 90%;
    margin: auto;
    overflow: hidden;
}

table {
    width: 100%;
    margin: 20px 0;
    border-collapse: collapse;
    white-space: nowrap;
    overflow-x: auto;
}

table, th, td {
    border: 1px solid #ddd;
}

th, td {
    padding: 8px;
    text-align: center;
}

th {
    background-color: #f2f2f2;
    color: #333;
}

tr:nth-child(even) {
    background-color: #f9f9f9;
}

tr:hover {
    background-color: #f1f1f1;
}

button {
    padding: 10px 20px;
    font-size: 16px;
    margin: 10px;
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    border-radius: 5px;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #0056b3;
}

button:focus {
    outline: none;
}

img {
    max-width: 80px;
    max-height: 80px;
    display: block;
    margin: auto;
}
```

XII-D. *Explicación del Código*

- **index.php**: La página de inicio proporciona una interfaz simple con un botón que lleva a la página que muestra la tabla de empleados.
- **basedatos.php**: Esta página se conecta a la base de datos PostgreSQL, obtiene los datos de los empleados, permite filtrar por número de empleado, y muestra los resultados en una tabla HTML.
- **styles.css**: Este archivo CSS define el estilo de la página, incluyendo la apariencia de la tabla y los botones, y asegura una presentación clara y profesional de los datos.

A continuación se muetran imagenes del funcionamiento de la vista grafica:

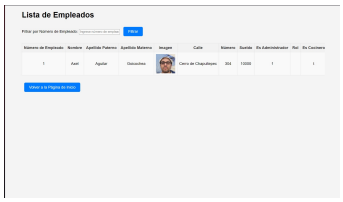


Figura 4. PHP

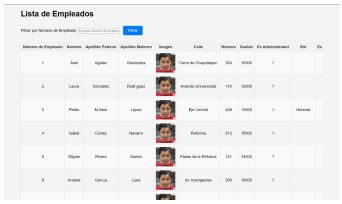


Figura 5. PHP

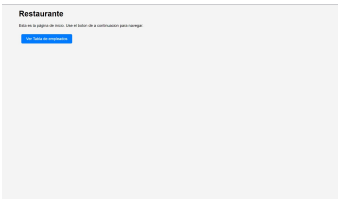


Figura 6. PHP

Esta implementación en PHP permite gestionar y visualizar los datos de los empleados de manera eficiente, proporcionando una interfaz simple para filtrar y consultar la información almacenada en la base de datos PostgreSQL.

XIII. Simulación de una Tabla de Base de Datos en Java

En esta sección, se presenta la implementación de un objeto en Java que simula el comportamiento de una tabla de base de datos. Este objeto permite realizar las operaciones básicas de inserción, borrado, actualización y consulta de información, replicando la funcionalidad esencial de una base de datos en un entorno de programación orientado a objetos. El código se ha dividido en tres archivos principales: `Main.java`, `Transacciones.java` y `Ordenes.java`.

XIII-A. Archivo `Main.java`

El archivo `Main.java` contiene el punto de entrada del programa y se encarga de inicializar las estructuras necesarias y ejecutar las operaciones simuladas en la tabla de facturas.

```
// Paquete principal
package IMPLEMENTACION;

import java.util.HashMap;
import java.util.Map;
import java.sql.Timestamp;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Inicializar la tabla de  rdenes  como un HashMap
        Map<String, Orden> tablaOrden = new HashMap<>();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            // Mostrar men  de opciones al usuario
            mostrarMenu();

            int opcion = scanner.nextInt();
            scanner.nextLine(); // Consumir el salto de l nea

            switch (opcion) {
                case 1:
                    // Insertar una nueva orden
                    insertarOrden(tablaOrden, scanner);
                    break;
                case 2:
                    // Seleccionar y mostrar todas las  rdenes
                    seleccionarOrdenes(tablaOrden);
                    break;
                case 3:
                    // Actualizar una orden existente
                    actualizarOrden(tablaOrden, scanner);
                    break;
                case 4:
                    // Eliminar una orden existente
                    eliminarOrden(tablaOrden, scanner);
                    break;
                case 5:
                    // Salir del programa
                    System.out.println("Saliendo del programa...");
                    return;
                default:
                    System.out.println("Opci n no v lida.");
            }
        }
    }

    // M todo para mostrar el men  de opciones
    private static void mostrarMenu() {
        System.out.println("\nMen :");
        System.out.println("1. Insertar orden");
        System.out.println("2. Seleccionar  rdenes ");
        System.out.println("3. Actualizar orden");
        System.out.println("4. Eliminar orden");
        System.out.println("5. Salir");
        System.out.print("Ingrese una opci n: ");
    }

    // M todo para insertar una nueva orden
    private static void insertarOrden(Map<String, Orden> tablaOrden, Scanner scanner) {
        System.out.print("Ingrese el folio de la orden: ");
        String folio = scanner.nextLine();

        System.out.print("Ingrese la fecha de la orden (YYYY-MM-DD HH:mm:ss): ");
        Timestamp fechaOrden = Timestamp.valueOf(scanner.nextLine());

        System.out.print("Ingrese el total de la orden: ");
        double total = scanner.nextDouble();

        System.out.print("Ingrese el n mero de empleado: ");
        int numEmpleado = scanner.nextInt();
        scanner.nextLine(); // Consumir el salto de l nea

        Orden orden = new Orden(folio, fechaOrden, total, numEmpleado);
        Transaccion transaccionInsertar = new Transaccion(tablaOrden, orden, "INSERT");
        Thread hiloInsertar = new Thread(transaccionInsertar);
        hiloInsertar.start();

        try {
            hiloInsertar.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // M todo para seleccionar y mostrar todas las  rdenes
```

```
    private static void seleccionarOrdenes(Map<String, Orden> tablaOrden) {
        if (tablaOrden.isEmpty()) {
            System.out.println("No hay   rdenes registradas.");
            return;
        }

        System.out.println("\nOrdenes:");
        for (Orden orden : tablaOrden.values()) {
            System.out.println(orden);
        }
    }

    // M todo para actualizar una orden existente
    private static void actualizarOrden(Map<String, Orden> tablaOrden, Scanner scanner) {
        System.out.print("Ingrese el folio de la orden a actualizar: ");
        String folio = scanner.nextLine();

        if (!tablaOrden.containsKey(folio)) {
            System.out.println("Error: Orden no existe.");
            return;
        }

        Orden orden = tablaOrden.get(folio);

        System.out.print("Ingrese el nuevo total de la orden: ");
        double nuevoTotal = scanner.nextDouble();
        scanner.nextLine();

        orden.setTotal(nuevoTotal);

        Transaccion transaccionActualizar = new Transaccion(tablaOrden, orden, "UPDATE");
        Thread hiloActualizar = new Thread(transaccionActualizar);
        hiloActualizar.start();

        try {
            hiloActualizar.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // M todo para eliminar una orden existente
    private static void eliminarOrden(Map<String, Orden> tablaOrden, Scanner scanner) {
        System.out.print("Ingrese el folio de la orden a eliminar: ");
        String folio = scanner.nextLine();

        if (!tablaOrden.containsKey(folio)) {
            System.out.println("Error: Orden no existe.");
            return;
        }

        Orden orden = tablaOrden.get(folio);

        Transaccion transaccionEliminar = new Transaccion(tablaOrden, orden, "DELETE");
        Thread hiloEliminar = new Thread(transaccionEliminar);
        hiloEliminar.start();

        try {
            hiloEliminar.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

XIII-B. Archivo `Transacciones.java`

El archivo `Transacciones.java` define la clase `Transaccion` que se encarga de realizar las operaciones de inserción, actualización y eliminación de órdenes de forma concurrente.

```
// Paquete principal
package IMPLEMENTACION;
import java.util.Map;

// Clase Transaccion
public class Transaccion implements Runnable {
    private Map<String, Orden> tablaOrden;
    private Orden orden;
    private String operacion;

    // Constructor de la clase Transaccion
    public Transaccion(Map<String, Orden> tablaOrden, Orden orden, String operacion) {
        this.tablaOrden = tablaOrden;
        this.orden = orden;
        this.operacion = operacion;
    }

    @Override
    public void run() {
        synchronized (tablaOrden) {
            switch (operacion) {
                case "INSERT":
                    insertarOrden();
                    break;
                case "UPDATE":
                    actualizarOrden();
                    break;
                case "DELETE":
                    eliminarOrden();
                    break;
            }
        }
    }

    // M todo para insertar una orden en la tabla
    private void insertarOrden() {
        if (tablaOrden.containsKey(orden.getFolio())) {
            System.out.println("Error: Folio ya existe");
            return;
        }
        if (!verificarEmpleado(orden.getNumEmpleado())) {
            System.out.println("Error: Empleado no existe");
            return;
        }
        tablaOrden.put(orden.getFolio(), orden);
        System.out.println("Orden insertada: " + orden.getFolio());
    }

    // M todo para actualizar una orden en la tabla
    private void actualizarOrden() {
        if (!tablaOrden.containsKey(orden.getFolio())) {
            System.out.println("Error: Folio no existe");
            return;
        }
```

```
        if (!verificarEmpleado(orden.getNumEmpleado())) {
            System.out.println("Error: Empleado no existe");
            return;
        }
        tablaOrden.put(orden.getFolio(), orden);
        System.out.println("Orden actualizada: " + orden.getFolio());
    }

    // M todo para eliminar una orden de la tabla
    private void eliminarOrden() {
        if (!tablaOrden.containsKey(orden.getFolio())) {
            System.out.println("Error: Folio no existe");
            return;
        }
        tablaOrden.remove(orden.getFolio());
        System.out.println("Orden eliminada: " + orden.getFolio());
    }

    // M todo para verificar la existencia de un empleado
    private boolean verificarEmpleado(int numEmpleado) {
        return true; // Aqu  asumimos que siempre existe
    }
}
```

XIII-C. *Archivo `Ordenes.java`*

El archivo `Ordenes.java` define la clase `Orden`, que encapsula los detalles de una orden y proporciona métodos para acceder y modificar sus atributos.

```
// Paquete principal
package IMPLEMENTACION;

import java.sql.Timestamp;
import java.util.regex.Pattern;

// Clase Orden
public class Orden {
    private String folio;
    private Timestamp fechaOrden;
    private double total;
    private int numEmpleado;

    // Constructor
    public Orden(String folio, Timestamp fechaOrden, double total, int numEmpleado) {
        if (!Pattern.matches(".*-.*", folio)) {
            throw new IllegalArgumentException("Folio debe tener el formato '%-%'");
        }
        this.folio = folio;
        this.fechaOrden = fechaOrden;
        this.total = total;
        this.numEmpleado = numEmpleado;
    }

    // Getters y setters
    public String getFolio() {
        return folio;
    }

    public void setFolio(String folio) {
        if (!Pattern.matches(".*-.*", folio)) {
            throw new IllegalArgumentException("Folio debe tener el formato '%-%'");
        }
        this.folio = folio;
    }

    public Timestamp getFechaOrden() {
        return fechaOrden;
    }

    public void setFechaOrden(Timestamp fechaOrden) {
        this.fechaOrden = fechaOrden;
    }

    public double getTotal() {
        return total;
    }

    public void setTotal(double total) {
        this.total = total;
    }

    public int getNumEmpleado() {
        return numEmpleado;
    }

    public void setNumEmpleado(int numEmpleado) {
        this.numEmpleado = numEmpleado;
    }

    @Override
    public String toString() {
        return "Orden{" +
                "folio='" + folio + '\'' +
                ", fechaOrden=" + fechaOrden +
                ", total=" + total +
                ", numEmpleado=" + numEmpleado +
                '}';
    }
}
```

XIII-D. *Explicación de las Operaciones*

- **Crear (Insertar)**: El método `insertarOrden()` en la clase `Transaccion` agrega una nueva orden a la tabla de órdenes.
- **Leer (Consultar)**:
  - El método `seleccionarOrdenes()` en la clase `Main` devuelve todas las órdenes almacenadas.
  - El método `getOrden()` puede implementarse en la clase `Orden` para devolver detalles de una orden específica.
- **Actualizar**: El método `actualizarOrden()` en la clase `Transaccion` actualiza una orden existente en la tabla.
- **Borrar**: El método `eliminarOrden()` en la clase `Transaccion` elimina una orden de la tabla utilizando su folio.

Esta implementación en Java simula eficazmente las operaciones básicas de una tabla de base de datos, proporcionando una estructura clara para la gestión de datos de órdenes en un entorno orientado a objetos.

XIV. Desafíos y Aprendizajes

A lo largo del desarrollo de este proyecto, enfrentamos diversos desafíos y adquirimos valiosos aprendizajes que en-

riquecieron nuestra experiencia en el ámbito de las bases de datos y la gestión de proyectos.

XIV-A. Desafíos

- **Gestión del Tiempo:** Uno de los principales desafíos fue la gestión del tiempo. A medida que avanzábamos en el proyecto, nos encontramos con retrasos en los entregables debido a una planificación inicial deficiente. Esto nos llevó a trabajar a un ritmo acelerado en las etapas finales para cumplir con los plazos establecidos.
- **Herramientas de Desarrollo:** La falta de un manejador de base de datos colaborativo fue otra dificultad significativa. Cada miembro del equipo trabajaba en su propia base de datos "personal", lo que complicó la integración y sincronización del trabajo. Implementamos un repositorio de GitHub para administrar el código y las versiones, lo cual ayudó a mitigar este problema.
- **Reajustes en el Diseño:** Durante el diseño lógico, nos dimos cuenta de que algunos aspectos no estaban contemplados adecuadamente, lo que nos obligó a retroceder y revisar fases anteriores. Este proceso de revisión y corrección fue crucial para asegurar la congruencia y calidad del diseño final.

XIV-B. Aprendizajes

- **Aplicación Práctica de Conocimientos:** El proyecto nos permitió aplicar los conocimientos adquiridos en el curso, desde el análisis de requisitos hasta la implementación en SQL. Diseñamos modelos entidad-relación y relacionales, creando una base de datos funcional que cumplió con los objetivos establecidos.
- **Trabajo en Equipo:** Aprendimos la importancia de la comunicación y la colaboración en equipo. Mantener una comunicación constante nos permitió gestionar cambios y avances de manera eficiente. La distribución de tareas, aunque no siempre equilibrada, mejoró significativamente hacia el final del proyecto.
- **Desarrollo de Habilidades Técnicas:** La implementación de triggers e índices nos demostró cómo mejorar la integridad y el rendimiento de las consultas. Estos elementos fueron esenciales para la optimización de nuestra base de datos y reflejaron la aplicación directa de las teorías aprendidas en el curso.
- **Planificación y Flexibilidad:** Este proyecto nos enseñó la importancia de una buena planificación y la necesidad de ser flexibles y adaptarnos a los cambios. La gestión de proyectos es un proceso dinámico que requiere ajustes continuos y una visión clara de los objetivos.

En resumen, este proyecto no solo nos permitió consolidar nuestros conocimientos teóricos, sino también desarrollar habilidades prácticas y de gestión que serán valiosas en nuestro futuro profesional. A pesar de los desafíos encontrados, logramos completar todas las etapas propuestas y adquirimos una experiencia significativa en el campo de las bases de datos.

XV. Conclusiones

XV-A. Cruz Miranda Luis Eduardo

De acuerdo con el objetivo establecido para este proyecto, se puede concluir que este se llevó a cabo en su totalidad y de manera satisfactoria. Para llevar a cabo el desarrollo de nuestra solución se realizó un análisis profundo de los requisitos del problema; una vez realizados se comenzó con los modelos y diseños de solución del problema, los cuales no tuvieron dificultades algunas. Cuando dicho análisis se comenzó a implementar, a causa de una gestión de tiempo carente, nuestros entregables comenzaron a aplazarse, por consiguiente, el desarrollo de la solución propuesta tuvo dificultades como el trabajo a una mayor marcha. Pese a estas dificultades se logró completar con cada una de las etapas propuestas y visualizadas. El desarrollo de este proyecto nos permitió poner a prueba los conocimientos previamente adquiridos y por su puesto mejoró nuestra lógica y forma de afrontar problemas de este tipo

XV-B. De la Rosa Lara Gustavo

El desarrollo de un proyecto partiendo desde un requerimiento siempre tiene sus particularidades y sus problemas. En este caso hubieron muchos reajustes de avances que tomabamos por terminados, esto nos otorga una muestra de lo que una buena planificación en cuanto a plazos de tiempo puede hacer, lo cual en nuestro caso fue muy mejorable. La implementación haciendo una simulación de una tabla da buen entendimiento de lo que hace un manejador detrás de escena, y eso considerando la carrera que cursamos tiene bastante valor.

XV-C. Dunzz Llampallas Alan

En el presente proyecto se diseñó e implementó una base de datos de acuerdo a un escenario planteado en un restaurante. El diseño de la base comenzó con entender los requerimientos planteados por el cliente, extrayendo las entidades de las cuales se van a almacenar datos, sus atributos, relaciones y especificaciones adicionales. A partir de lo anterior se realizó un modelo entidad relación, que posteriormente fue convertido en un modelo relacional. Una vez teniendo dicho modelo, se creó la base de datos y funciones auxiliares con ayuda de pgadmin a través de comandos de SQL y a continuación se insertaron los datos correspondientes. Se realizaron también 2 implementaciones: un sitio web y un objeto en Java que asemeja a una base de datos. Como se puede observar, este proyecto abarcó todos los temas vistos a lo largo del curso y permitió aplicar todo el aprendizaje recabado en un caso de uso, permitiéndonos ampliar nuestro conocimiento y experiencia en el ámbito de las bases de datos.

XV-D. González Arredondo Rafael

Este proyecto ha abordado una amplia gama de temas que van más allá del contenido impartido en la materia. Desde conceptos avanzados hasta aplicaciones prácticas, hemos explorado áreas que no fueron completamente cubiertas en el curso, lo que ha resultado en un desafío emocionante y enriquecedor para nuestro equipo. Aunque enfrentamos

dificultades en la gestión del tiempo, lo que afectó la calidad final del proyecto, esta experiencia nos proporcionó valiosas lecciones sobre trabajo en equipo y aplicación práctica de conocimientos en el campo de las bases de datos. Trabajar juntos nos permitió desarrollar habilidades de colaboración y comunicación, mientras que la aplicación práctica nos brindó una comprensión más profunda de los conceptos teóricos. En última instancia, este proyecto ha sido una experiencia de aprendizaje significativa que nos ha permitido crecer tanto a nivel individual como colectivo. A pesar de los desafíos encontrados, valoramos las lecciones aprendidas y estamos motivados para aplicarlas en futuros proyectos, con el objetivo de alcanzar un mayor éxito y excelencia en nuestro trabajo.

XV-E. Lemus Gonzales Javier Isaac

Considero que el producto final fue satisfactorio, ya que la mayoría de las fases de desarrollo fueron realizadas en tiempo y forma. Concluyo que la distribución de trabajo a pesar de no ser perfecta, fue buena. Por momentos la carga de trabajo se distribuía de manera desbalanceada, pero en la fase final considero que se compensó.

Opino que las principales dificultades que se nos presentaron fueron:

La herramienta para el desarrollo de la BD, ya que no contamos con un manejador colaborativo. Sin embargo salimos a flote, ya que implementamos un repositorio de GitHub para administrar el código y sus versiones a pesar de que cada uno tuviera que ir actualizando su base ”personal”.

A pesar de realizar la fase de analisis a profundidad, a la hora del diseño lógico ciertas cosas no estaban contempladas de la mejor manera. Lo que nos obligó a retroceder en las fases y partir nuevamente desde donde se necesitara. Al final, logramos un buen diseño y una buena implementación, generando congruencia entre cada fase con la anterior.

Los aciertos que tuvimos fueron:

Buena comunicación, ya que todos los miembros del equipo estuvimos en comunicación todo el tiempo de desarrollo. Lo que nos permitió gestionar los cambios o analizar como se iba avanzando en el proyecto.

Buen nivel de conocimiento general de la materia, realmente fueron pocas las cosas que se investigaron por aparte. En las sesiones de teoría aprendimos gran parte de lo requerido. Lo que aceleró el trabajo, especialmente hasta antes del diseño físico.

XV-F. Moreno Santoyo Mariana

Este proyecto ha servido como una valiosa aplicación práctica de los conceptos aprendidos en el curso sobre bases de datos, facilitando una comprensión más profunda del diseño y manejo de bases de datos con PostgreSQL. Desde el desarrollo inicial de modelos Entidad-Relación hasta la creación de modelos relacionales intermedios y su implementación final en SQL, cada etapa ha reforzado nuestro conocimiento teórico con experiencia práctica. Este proceso ha permitido no solo visualizar sino también ejecutar la transformación de diagramas conceptuales en estructuras de base de datos funcionales y optimizadas. Los triggers y índices implementados han demostrado ser herramientas esenciales para mejorar la integridad y el rendimiento de las consultas, ilustrando la aplicación directa de teorías del curso en soluciones tecnológicas reales.