


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Bases de Datos
Proyecto Final

ba

Aguilar Martínez Erick Yair

Professor:
Ing. Fernando Arreola
Semestre: 2024-1

26 de noviembre de 2023

1. Proposito

Desarrollar una aplicación de gestión integral para un restaurante que se conecte con una base de datos, garantizando el almacenamiento y acceso a la información. La aplicación deberá ofrecer una experiencia de usuario fluida al incorporar todas las funcionalidades necesarias que cumplan con los requerimientos.Con todo la anterior se demostrará la aplicación de los conocimientos adquiridos en el ámbito de las bases de datos.

2. Alcance

Este proyecto va a transformar por completo la forma en que opera el restaurante, mejorando significativamente la eficiencia del negocio al no perder tiempo en tedioso papelo y generación de reportes. A medida que avanzamos hacia la era digital, reconocemos la necesidad de un sistema digital robusto y eficaz que no solo simplifique nuestras operaciones internas, sino que también mejore la experiencia de los clientes con tiempos de respuesta aún mas cortos.

2.1. Alcance del Producto

El proyecto abarca el desarrollo de un sistema digital, con un enfoque particular en la implementación de una base de datos. Mi objetivo es centralizar y organizar la información crucial que se genera en el restaurante, desde detalles de empleados y menú hasta registros de ventas. Al hacerlo, espero maximizar las operaciones, reducir errores y generar un servicio más rápido.

2.2. Valor del Producto

La importancia de este producto radica en su capacidad para revolucionar la forma de trabajar. Al consolidar datos vitales en una plataforma digital, permitirá una toma de decisiones más informada.

2.3. Público Objetivo

El público objetivo abarca al equipo interno. Desde su perspectiva, este sistema será una herramienta indispensable para gestionar las tendencias del negocio.

2.4. Uso Previsto

Anticipamos que el equipo interno utilizará el sistema para realizar un seguimiento de las existencias y analizar datos de ventas.

3. Análisis del Problema

Naturalmente es necesario comenzar con la captura de los requerimientos tanto funcionales con no funcionales mediante artefactos como:

- **Entrevistas con los interesados:** Realizar entrevistas con los usuarios, clientes y otras partes interesadas para comprender sus necesidades y expectativas con respecto al software.
- **Encuestas y cuestionarios:** Diseñar encuestas y cuestionarios para recopilar información de un gran número de personas y analizar sus respuestas para identificar patrones y tendencias.
- **Talleres de trabajo (Workshops):** Organizar talleres interactivos con representantes de los usuarios y el equipo de desarrollo para discutir y definir los requerimientos de manera colaborativa.
- **Observación del usuario:** Observar cómo los usuarios interactúan con sistemas existentes o prototipos para comprender su flujo de trabajo y necesidades específicas.
- **Prototipado:** Desarrollar prototipos rápidos del software para que los usuarios puedan visualizar y probar las funcionalidades, lo que ayuda a identificar y refinar los requerimientos.
- **Análisis de documentos existentes:** Revisar documentos como manuales, informes y otros registros para extraer información relevante sobre los procesos y requerimientos del sistema.

Sin embargo, esto no forma parte del objetivo; por lo tanto, el Ing. Fernando Arreola ha proporcionado un resumen que será suficiente para comprender las necesidades del restaurante. El resto del trabajo para esta etapa consiste en especificar, mediante diagramas, los requerimientos y clasificarlos, incluyendo diagramas de casos de uso, entre otros.

3.1. Perspectiva del Producto

El sistema se posiciona como una solución determinante para mejorar significativamente la operativa y la experiencia del cliente. Se espera que el producto se integre sin problemas en el entorno actual del restaurante, aprovechando tecnologías web para proporcionar una plataforma robusta y fácil de usar desde la perspectiva del restaurante y sus usuarios.

3.2. Funciones del Producto

3.2.1. Panel de Control

El sistema debe proporcionar un panel de control que permita visualizar los siguientes datos:

1. **Ingresos del mes:** El total de ingresos generados en el restaurante durante el mes en curso(fig.1).

2. **Horas pico:** Identificación de las horas del día en las que se registran más órdenes(fig.1).

3.2.2. Estadísticas y Reportes

1. El sistema debe permitirá visualizar informes que incluyan datos como las ventas del mes, el platillo más vendidos y los ingresos generados(fig.1).

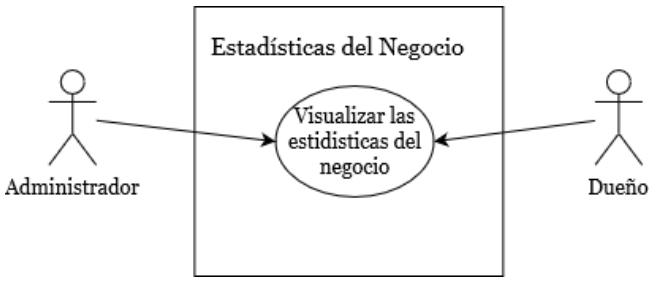

Figura 1: Diagrama de casos de uso para las Estadísticas del Negocio

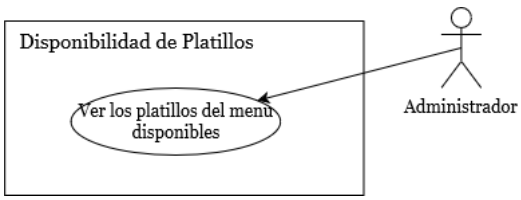

Figura 2: Diagrama de Casos de Uso para el Manejo del Menú

3.3. Requerimientos no Funcionales

3.3.1. Usabilidad

1. La interfaz de usuario en la plataforma web debe ser intuitiva y fácil.

2. La respuesta del sistema a las acciones del usuario debe ser instantánea, proporcionando una experiencia sin demoras ni interrupciones perceptibles.

3.3.2. Rendimiento

1. El sistema debe ser capaz de manejar un alto volumen de transacciones simultáneas, especialmente durante las horas pico del restaurante, sin degradación significativa en el rendimiento.

3.3.3. Disponibilidad

1. El sistema debe estar disponible en los horarios en que el restaurante este operando para garantizar que los empleados puedan acceder a la información necesaria en cualquier momento.

3.3.4. Seguridad

1. Los datos almacenados en la base de datos, incluyendo información de empleados y clientes, deben estar protegidos mediante medidas de seguridad robustas, como encriptación de datos y autenticación de usuarios.

3.3.5. Escalabilidad

1. El sistema debe ser escalable para permitir la adición de nuevos empleados, productos y clientes sin afectar el rendimiento general.

2. Debe ser posible expandir la capacidad de almacenamiento y procesamiento de la base de datos para adaptarse al crecimiento futuro del restaurante y su clientela.

3.3.6. Integración

1. El sistema debe ser compatible con otros sistemas de software utilizados en el restaurante, como sistemas de contabilidad y sistemas de gestión de inventario, para facilitar la transferencia de datos y garantizar la coherencia en toda la operación.

Estos requerimientos no funcionales son esenciales para garantizar que el sistema no solo cumpla con las expectativas funcionales, sino que también sea seguro, fácil de usar, confiable y capaz de adaptarse a las demandas cambiantes del entorno.

4. Plan de Trabajo

4.1. Selección de las Herramientas

En esta fase, se seleccionaran las tecnológicas necesarias para agilizar el desarrollo del proyecto. Cada una tendrá una razón en función de su idoneidad para los requisitos y objetivos específicos del sistema.

4.2. Configuración de la Base de Datos

Se realizará la configuración inicial de la base de datos, abarcando la selección del sistema de gestión de bases de datos (PostgreSQL), la elección de la imagen de Docker, la configuración de variables de entorno, la creación de esquemas y la definición de relaciones entre entidades. Todo esto se llevará a cabo con el objetivo de satisfacer las necesidades de almacenamiento de datos del sistema.

4.3. Implementación de Scripts Iniciales de la Base de Datos

Se implementaron scripts iniciales en la base de datos con el fin de establecer la estructura básica y los datos iniciales. Esto proporciona la base necesaria para el desarrollo y las pruebas iniciales del sistema, además de facilitar un enfoque iterativo en el proceso de desarrollo.

4.4. Desarrollo de la Interfaz Gráfica

Este paso involucrará la creación de diseños e implementación de componentes visuales que aseguren una experiencia de usuario intuitiva esperando generar un atractivo visual.

4.5. Desarrollo de la Aplicación Backend

Se tiene planeado desarrollar la aplicación backend para gestionar la lógica de negocio y facilitar la comunicación con la base de datos. Se implementarán las funciones y servicios necesarios para garantizar el cumplimiento de las funciones especificadas anteriormente.

4.6. Comunicación entre los Módulos Clientes y Backend

Para esta ultima fase habrá que implementar protocolos de comunicación, gestionar solicitudes y respuestas, y garantizar un flujo de información coherente entre las diferentes partes del sistema (Cliente y Servidor).

5. Diseño

En esta sección del documento, se abordarán las fases de diseño de la base de datos,y como su nombre lo dice será el funcionamiento coherente del sistema. El proceso de diseño se ha dividido en dos partes: el Modelo Relacional y el Modelo Físico. Cada fase desempeña un papel en la estructuración y optimización de la base de datos para satisfacer los requisitos específicos del proyecto.

5.1. Modelo Relacional

En la fase del Modelo Relacional, se ha trabajado en la representación lógica de la base de datos. Se han identificado las entidades clave y sus relaciones, y se ha diseñado un esquema que refleje sin ambigüedades la estructura de la información. Este modelo proporciona una base conceptual para la implementación de la base de datos, definiendo cómo se relacionan entre sí las diferentes entidades y cómo se gestionarán los datos.

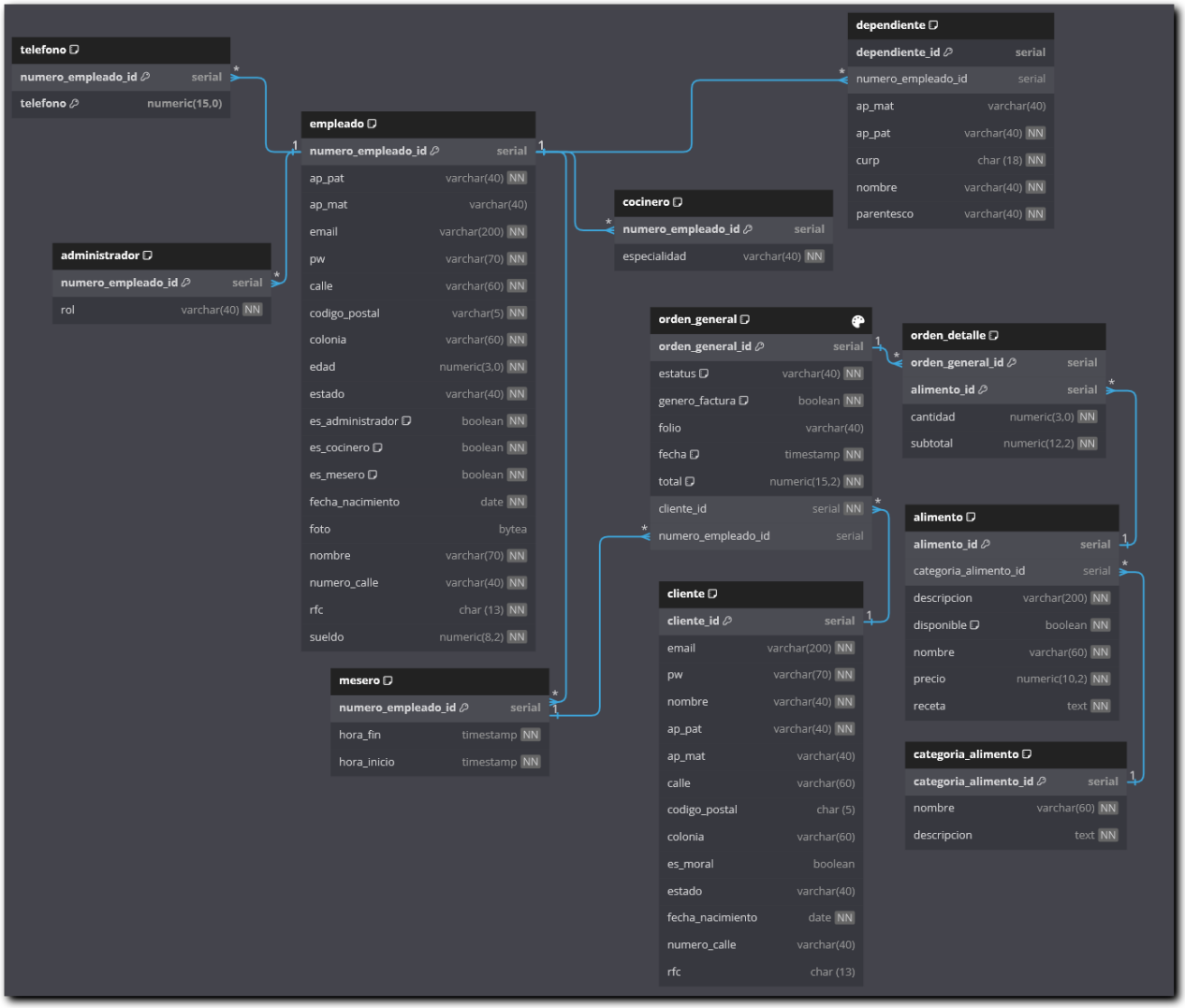

Figura 3: Diagrama Entidad Relación. Para mas detalle ver el siguiente Link

```sql
CREATE TABLE
    empleado(
        numero_empleado_id serial,
        ap_pat varchar(40) not null,
        ap_mat varchar(40),
        email varchar(200) not null,
        pw varchar(70) not null,
        calle varchar(60) not null,
        codigo_postal varchar(5) not null,
        colonia varchar(60) not null,
        edad numeric(3, 0) not null,
        estado varchar(40) not null,
        es_administrador boolean default false not null,
        es_cocinero boolean default false not null,
        es_mesero boolean default false not null,
        fecha_nacimiento date not null,
        foto bytea null,
        nombre varchar(70) not null,
        numero_calle varchar(40) not null,
        rfc char(13) not null,
        sueldo numeric(8, 2) not null,
        constraint empleado_pk primary key(numero_empleado_id),
        constraint empleado_rfc_uk unique(rfc),
        constraint subtipo_empleado_total_chk check( (es_administrador = true)
            or (es_cocinero = true)
            or (es_mesero = true)
        ),
        constraint empleado_codigo_postal_chk check(codigo_postal ~ '^[0-9]{5}$'),
        constraint empleado_edad_chk check(edad >= 0),
        constraint empleado_email_uk unique(email),
        constraint empleado_email_chk check(email ~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$')
    );
```

Figura 4: Empleado

5.2. Modelo Físico

La fase del Modelo Lógico se centra en la implementación concreta para un manejador de bases de datos especifico. Se han tomado decisiones específicas sobre la implementación de índices, claves primarias y foráneas, así como la optimización de la estructura para el rendimiento. Se han considerado aspectos de almacenamiento y acceso a los datos para garantizar la operatividad de la base de datos en la aplicación práctica del sistema.

```
create table
    administrador(
        numero_empleado_id serial,
        rol varchar(40) not null,
        constraint administrador_pk primary key(numero_empleado_id),
        constraint administrador_numero_empleado_id_fk foreign key(numero_empleado_id) references empleado(numero_empleado_id),
        constraint administrador_rol_chk check(
            rol in (
                'Dueño',
                'Gerente General',
                'Gerente de Operaciones',
                'Gerente de Recursos Humanos',
                'Gerente Financiero',
                'Gerente de Marketing'
            )
        )
    );
```

Figura 5: Administrador

```
create table
    cliente(
        cliente_id serial,
        email varchar(200) not null,
        pw varchar(70) not null,
        nombre varchar(40) not null,
        ap_pat varchar(40) not null,
        ap_mat varchar(40),
        calle varchar(60),
        codigo_postal char(5),
        colonia varchar(60),
        es_moral boolean,
        estado varchar(40),
        fecha_nacimiento date not null,
        numero_calle varchar(40),
        rfc char(13),
        constraint cliente_pk primary key(cliente_id),
        constraint cliente_rfc_uk unique(rfc),
        constraint cliente_email_uk unique(email),
        constraint cliente_email_chk check(email ~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$')
    );
```

Figura 6: Cliente

```
create table
    dependiente(
        dependiente_id serial,
        numero_empleado_id serial,
        ap_mat varchar(40),
        ap_pat varchar(40) not null,
        curp char(18) not null,
        nombre varchar(40) not null,
        parentesco varchar(40) not null,
        constraint dependiente_pk primary key(dependiente_id),
        constraint dependiente_numero_empleado_id_fk foreign key(numero_empleado_id) references empleado(numero_empleado_id),
        constraint dependiente_curp_uk unique(curp),
        constraint dependiente_parentesco_chk check(
            parentesco in (
                'Hermano',
                'Hermana',
                'Padre',
                'Madre',
                'Hijo',
                'Hija',
                'Abuelo',
                'Abuela',
                'Nieto',
                'Nieta',
                'Tío',
                'Tía',
                'Primo',
                'Prima',
                'Sobrino',
                'Sobrina'
            )
        )
    );
```

Figura 7: Dependiente

```
create table
    mesero(
        numero_empleado_id serial,
        hora_fin timestamp not null,
        hora_inicio timestamp not null,
        constraint mesero_pk primary key(numero_empleado_id),
        constraint mesero_numero_empleado_id_fk foreign key(numero_empleado_id) references empleado(numero_empleado_id),
        constraint mesero_hora_inicio_chk check(hora_inicio < hora_fin)
    );
```

Figura 8: Mesero

```sql
create table
    cocinero(
        numero_empleado_id serial,
        especialidad varchar(40) not null,
        constraint cocinero_pk primary key(numero_empleado_id),
        constraint cocinero_numero_empleado_id_fk foreign key(numero_empleado_id) references empleado(numero_empleado_id),
        constraint cocinero_especialidad_chk check(
            especialidad in (
                'Cocina Francesa',
                'Cocina Italiana',
                'Cocina Japonesa',
                'Cocina Mexicana',
                'Repostería',
                'Cocina Vegetariana',
                'Cocina de Fusión',
                'Parrilla y Asados',
                'Cocina Tailandesa',
                'Cocina Molecular'
            )
        )
    );
```

Figura 9: Cocinero

```sql
create table
    categoria_alimento(
        categoria_alimento_id serial,
        nombre varchar(60) not null,
        descripcion text not null,
        constraint categoria_alimento_pk primary key(categoria_alimento_id)
    );
```

Figura 10: Categoría Alimento

```sql
create table
    alimento(
        alimento_id serial,
        categoria_alimento_id serial,
        descripcion varchar(200) not null,
        disponible boolean default false not null,
        nombre varchar(60) not null,
        precio numeric(10, 2) not null,
        receta text not null,
        constraint alimento_pk primary key(alimento_id),
        constraint alimento_categoria_alimento_id_fk foreign key(categoria_alimento_id) references categoria_alimento(categoria_alimento_id)
    );
```

Figura 11: Alimento

```sql
create table
    orden_general(
        orden_general_id serial,
        estatus varchar(40) default 'REGISTRADA' not null,
        genero_factura boolean default false not null,
        folio varchar(40),
        fecha timestamp default now() not null,
        total numeric(15, 2) DEFAULT 0 not null,
        cliente_id serial not null,
        numero_empleado_id serial,
        constraint orden_general_pk primary key(orden_general_id),
        constraint orden_general_cliente_id_fk foreign key(cliente_id) references cliente(cliente_id),
        constraint orden_general_numero_empleado_fk foreign key(numero_empleado_id) references mesero(numero_empleado_id),
        constraint orden_general_estatus_chk check(
            estatus in (
                'REGISTRADA',
                'EN PROGRESO',
                'ENTREGADA',
                'CANCELADA'
            )
        ),
        constraint orden_general_folio_chk check(folio ~ '^ORD-[1-9][0-9]*$')
    );
```

Figura 12: Orden General

```sql
create table
    orden_detalle(
        orden_general_id serial,
        alimento_id serial,
        cantidad numeric(3, 0) not null,
        subtotal numeric(12, 2) not null,
        constraint orden_detalle_pk primary key(orden_general_id, alimento_id),
        constraint orden_detalle_orden_general_id foreign key(orden_general_id) references orden_general(orden_general_id),
        constraint orden_detalle_alimento_id foreign key(alimento_id) references alimento(alimento_id),
        constraint orden_detalle_cantidad_chk check(cantidad > 0)
    );
```

Figura 13: Orden Detalle

```sql
CREATE OR REPLACE FUNCTION verificar_email()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT *
        FROM empleado
        WHERE email = NEW.email
    ) THEN
        RAISE EXCEPTION 'El email ya existe en la tabla empleado.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER cliente_verificar_email
BEFORE INSERT OR UPDATE
ON cliente
FOR EACH ROW
EXECUTE FUNCTION verificar_email();
```

Figura 14: Código fuente del Trigger Cliente Verificar Email

6. Implementación

En la fase de implementación, se han abordado diversos aspectos para traducir el diseño de la base de datos a una aplicación funcional. A continuación, se proporciona una breve descripción de cada uno de los elementos implementados:

6.1. Triggers

Los triggers son acciones automáticas que se ejecutan en respuesta a eventos específicos en la base de datos. Han sido implementados para realizar acciones predefinidas, como la actualización de datos o la validación de restricciones.

6.1.1. Cliente Verificar Email

Se ejecuta antes de realizar operaciones de inserción o actualización (`BEFORE INSERT OR UPDATE`) en la tabla `cliente`. El *trigger* invoca una función llamada `verificar_email()` que tiene la tarea de verificar si el correo electrónico (campo `email`) que se está intentando insertar o actualizar ya existe en la tabla `empleado`. Si existe, se lanza una excepción indicando que el correo electrónico ya existe en la tabla `empleado`. Si no existe, la función simplemente devuelve la nueva fila (`NEW`).

```sql
CREATE OR REPLACE FUNCTION verificar_email_empleado()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM cliente
        WHERE email = NEW.email
    ) THEN
        RAISE EXCEPTION 'El email ya existe en la tabla cliente.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER empleado_verificar_email
BEFORE INSERT OR UPDATE
ON empleado
FOR EACH ROW
EXECUTE FUNCTION verificar_email_empleado();
```

Figura 15: Código fuente del Trigger Empleado Verificar Email

6.1.2. Empleado Verificar Email

Se ejecuta antes de realizar operaciones de inserción o actualización (`BEFORE INSERT OR UPDATE`) en la tabla `empleado`. El *trigger* invoca una función llamada `verificar_email_empleado()` que tiene la tarea de verificar si el correo electrónico (campo `email`) que se está intentando insertar o actualizar ya existe en la tabla `cliente`. Si existe, se lanza una excepción indicando que el correo electrónico ya existe en la tabla `cliente`. Si no existe, la función simplemente devuelve la nueva fila (`NEW`).

6.1.3. Trigger Validar Alimento

Se ejecuta antes de realizar operaciones de inserción (`BEFORE INSERT`) en la tabla `orden_detalle`. Este *trigger* invoca una función llamada `validar_alimento_disponible()` que tiene la tarea de verificar si el alimento seleccionado para la nueva fila en la tabla `orden_detalle` está disponible. Si el alimento no está disponible, se lanza una excepción indicando que el alimento seleccionado no está disponible. Si el alimento está disponible, la función simplemente devuelve la nueva fila (`NEW`).

```sql
CREATE OR REPLACE FUNCTION validar_alimento_disponible()
RETURNS TRIGGER AS $$
BEGIN
  IF (SELECT disponible FROM alimento WHERE alimento_id = NEW.alimento_id) <> true THEN
    RAISE EXCEPTION 'El alimento seleccionado no está disponible';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_validar_alimento
BEFORE INSERT ON orden_detalle
FOR EACH ROW
EXECUTE FUNCTION validar_alimento_disponible();
```

Figura 16: Código fuente del Trigger Validar Alimento

6.1.4. Actualizar Total Orden Detalle

Se ejecuta después de realizar operaciones de inserción, actualización o eliminación (`AFTER INSERT OR UPDATE OR DELETE`) en la tabla `orden_detalle`. Este *trigger* invoca una función llamada `actualizar_total_orden_general()` que tiene la tarea de recalcular y actualizar el campo `total` en la tabla `orden_general` basándose en la suma de los subtotales de las filas en la tabla `orden_detalle` asociadas a la misma orden general.

6.1.5. Autogenerar Folio

Se ejecuta antes de realizar operaciones de inserción (`BEFORE INSERT`) en la tabla `orden_general`. El *trigger* invoca una función llamada `generar_folio()` que tiene la tarea de asignar un valor al campo `folio` en la fila que se está insertando en la tabla `orden_general`. El valor asignado al campo `folio` está compuesto por la cadena 'ORD-' seguida del identificador de la orden general (`orden_general_id`).

```sql
CREATE OR REPLACE FUNCTION actualizar_total_orden_general()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE orden_general
    SET total = (
        SELECT SUM(subtotal)
        FROM orden_detalle
        WHERE orden_detalle.orden_general_id = NEW.orden_general_id
    )
    WHERE orden_general.orden_general_id = NEW.orden_general_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER actualizar_total_orden_detalle
AFTER INSERT OR UPDATE OR DELETE
ON orden_detalle
FOR EACH ROW
EXECUTE FUNCTION actualizar_total_orden_general();
```

Figura 17: Código fuente del Trigger Actualizar Total Orden Detalle

```sql
CREATE OR REPLACE FUNCTION generar_folio()
RETURNS TRIGGER AS $$
BEGIN
    NEW.folio := 'ORD-' || NEW.orden_general_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER autogenerar_folio
BEFORE INSERT ON orden_general
FOR EACH ROW
EXECUTE FUNCTION generar_folio();
```

Figura 18: Código fuente del Trigger Autogenerar Folio

```sql
CREATE OR REPLACE FUNCTION OBTENER_ORDENES_MESERO(NUMERO_EMPLEADO_PARAM
INT) RETURNS TABLE(CANTIDAD_ORDENES BIGINT, TOTAL_PAGADO
NUMERIC) AS $$
    $$ DECLARE es_mesero_boolean BOOLEAN;
    BEGIN
    SELECT
        es_mesero INTO es_mesero_boolean
    FROM empleado
    WHERE
        numero_empleado_id = numero_empleado_param;
    IF NOT es_mesero_boolean THEN RAISE EXCEPTION 'El empleado con número % no es un mesero.',
    numero_empleado_param;
    END IF;
    RETURN QUERY
    SELECT
        count(*) as cantidad_ordenes,
        COALESCE(sum(og.total), 0) as total_pagado
    FROM orden_general og
    WHERE
        og.numero_empleado_id = numero_empleado_param
        and to_char(og.fecha, 'YYYY-MM-DD') = to_char(CURRENT_DATE, 'YYYY-MM-DD');
    END;
    $$ LANGUAGE plpgsql;
```

Figura 19: Código fuente de la Función Obtener Ordenes Mesero

6.2. Funciones

Las funciones son bloques de código que devuelven un valor específico. Se han implementado funciones para realizar cálculos o procesamientos específicos en la base de datos y proporcionar resultados útiles para consultas y operaciones.

6.2.1. Obtener Ordenes Mesero

Toma un número de empleado como entrada y devuelve la cantidad de órdenes realizadas por ese empleado (si es un mesero) y el total pagado en el día actual. La función primero verifica si el empleado con el número proporcionado es un mesero. Si no es un mesero, la función lanza una excepción. Si es un mesero, la función realiza una consulta para obtener la cantidad de órdenes y el total pagado por ese mesero en el día actual, y luego devuelve estos resultados como un conjunto de resultados con las columnas `cantidad_ordenes` y `total_pagado`.

6.2.2. Obtener Estadísticas Ventas

Toma dos fechas como entrada y devuelve estadísticas de ventas en forma de cantidad total de ventas (`total_ventas`) y monto total vendido (`monto_total`). Si solo se proporciona una fecha (`fecha_inicio_param`), la función devuelve estadísticas para ese día. Si se proporciona un rango de fechas (`fecha_inicio_param` y `fecha_fin_param`), la función devuelve estadísticas para ese intervalo de tiempo.

```sql
CREATE OR REPLACE FUNCTION obtener_estadisticas_ventas(
    fecha_inicio_param DATE,
    fecha_fin_param DATE DEFAULT NULL
)
RETURNS TABLE (
    total_ventas BIGINT,
    monto_total NUMERIC
)
AS $$
BEGIN
    IF fecha_fin_param IS NULL THEN
        RETURN QUERY
        SELECT
            COUNT(orden_general_id) AS total_ventas,
            COALESCE(SUM(total), 0) AS monto_total
        FROM
            orden_general
        WHERE
            DATE_TRUNC('day', fecha) = DATE_TRUNC('day', fecha_inicio_param);
    ELSE
        RETURN QUERY
        SELECT
            COUNT(orden_general_id) AS total_ventas,
            COALESCE(SUM(total), 0) AS monto_total
        FROM
            orden_general
        WHERE
            fecha BETWEEN fecha_inicio_param AND fecha_fin_param;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Figura 20: Código fuente de la Función Obtener Estadísticas Ventas

6.3. Views

Las vistas (Views) son consultas almacenadas que actúan como tablas virtuales. Se han creado vistas para simplificar la visualización de datos complejos al proporcionar una capa de abstracción sobre las relaciones y la estructura de la base de datos. Las vistas mejoran la legibilidad y facilitan la recuperación de información.

6.3.1. Vista Platillo Mas Vendido

Muestra información sobre el platillo más vendido. La vista se crea a partir de la unión de tres tablas: `alimento`, `categoria_alimento`, y `orden_detalle`.

- La vista incluye columnas como `alimento_id`, `nombre_alimento`, `descripcion_alimento`, `precio_alimento`, `nombre_categoria`, y `cantidad_vendida`.
- Se unen las tablas `alimento`, `categoria_alimento`, y `orden_detalle` para obtener información detallada sobre los platillos, sus categorías y las órdenes asociadas.
- La cantidad de veces que un platillo específico ha sido vendido se cuenta utilizando la función `COUNT(od.alimento_id)`.
- Los resultados se agrupan por `alimento_id`, `nombre_alimento`, `descripcion_alimento`, `precio_alimento`, y `nombre_categoria` para obtener una fila por cada platillo.
- La vista se ordena en orden descendente según la cantidad de veces que se ha vendido cada platillo (`COUNT(od.alimento_id)`).
- Finalmente, se utiliza `LIMIT 1` para mostrar solo la fila del platillo más vendido.

6.4. Vista Factura Orden

Proporciona información para la generación de facturas de órdenes específicas.

- La vista incluye columnas como `nombre_completo`, `fecha`, `folio`, `importe`, `importe_iva`, y `fecha_vencimiento`.
- La información se extrae de las tablas `orden_general` y `cliente` mediante una unión (`JOIN`) basada en el identificador del cliente (`cliente_id`).
- La columna `nombre_completo` combina el nombre, apellido paterno y apellido materno del cliente.
- Se selecciona la fecha de la orden (`fecha`), el folio de la orden (`folio`), y el importe total de la orden (`importe`) desde la tabla `orden_general`.
- La columna `importe_iva` calcula el importe total más el 16 % de impuesto al valor agregado (IVA).
- La columna `fecha_vencimiento` calcula la fecha de vencimiento de la factura sumando tres días a la fecha de la orden.
- La vista filtra solo las órdenes que requieren generación de factura (`og.genero_factura = true`).

```sql
CREATE
OR REPLACE VIEW vista_platillo_mas_vendido AS
SELECT
    ad.alimento_id,
    ad.nombre AS nombre_alimento,
    ad.descripcion AS descripcion_alimento,
    ad.precio AS precio_alimento,
    ca.nombre AS nombre_categoria,
    COUNT(od.alimento_id) AS cantidad_vendida
FROM alimento ad
    JOIN categoria_alimento ca ON ad.categoria_alimento_id = ca.categoria_alimento_id
    JOIN orden_detalle od ON ad.alimento_id = od.alimento_id
GROUP BY
    ad.alimento_id,
    ad.nombre,
    ad.descripcion,
    ad.precio,
    ca.nombre
ORDER BY
    COUNT(od.alimento_id) DESC
LIMIT 1;
```

Figura 21: Código fuente de la Vista Platillo Mas Vendido

```sql
CREATE OR REPLACE VIEW vista_factura_orden AS
SELECT
    (c.nombre || ' ' || c.ap_pat || ' ' || c.ap_mat) as nombre_completo,
    og.fecha,
    og.folio,
    og.total as importe,
    (og.total * 1.16) as importe_iva,
    to_char((og.fecha + interval '3 days'),'YYYY-MM-DD') as fecha_vencimiento
FROM orden_general og
JOIN cliente c ON og.cliente_id = c.cliente_id
WHERE og.genero_factura = true;
```

Figura 22: Código fuente de la Vista Factura Orden

```sql
CREATE
OR REPLACE VIEW vista_productos_no_disponibles AS
SELECT
    alimento_id,
    nombre,
    descripcion,
    precio,
    disponible
FROM alimento
WHERE disponible = false;
```

Figura 23: Código fuente de la Vista Productos Disponibles

6.4.1. Vista Productos Disponibles

Muestra información acerca de los productos que no están disponibles. En términos sencillos, la vista selecciona las columnas `alimento_id`, `nombre`, `descripcion`, `precio`, y `disponible` de la tabla `alimento` y filtra las filas donde el valor de `disponible` es `false`.

7. Presentación Dashboard

Esta sencilla aplicación exhibe tres funciones que capacitan al usuario para verificar la información almacenada en la base de datos. Constituye una forma práctica de explorar el siguiente aspecto crucial de estos sistemas, dado que ahora procesamos y conferimos mayor significado y relevancia a dicha información para nuestro usuario final. Si tuviera que resumirlo en una frase, sería: "Dashboard actúa como una ventana intuitiva que potencia la comprensión y relevancia de los datos almacenados, proporcionando a los usuarios un acceso eficiente y significativo a la información existente dentro del sistema".

8. Conclusiones

Este proceso de desarrollo ha sido un fascinante y productivo viaje, donde he alcanzado metas significativas en un periodo corto. La oportunidad de aprender y aplicar nuevas habilidades durante este proyecto ha sido una experiencia enriquecedora, reforzando mi pasión por esta materia y brindándome la confianza necesaria para abordar desafíos con determinación. Me enorgullece el progreso alcanzado, y estoy emocionado por las perspectivas que este proyecto ha abierto para mi crecimiento continuo. Al reflexionar sobre experiencias previas, con simples aplicaciones en terminal valoro la evolución que he experimentado en mi carrera profesional.