Universidad Nacional Autónoma de México

Facultad De Ingeniería

Nombre de la materia: Bases de datos

Grupo: 1

Semestre 2022-2

Nombre del profesor: Ing.Fernando Arreola Franco

Nombre de los integrantes del equipo:

Andrés Urbano Andrea

Calvillo Martínez Erik Jonathan

De La Cruz Munguía Arely

Vázquez Pérez Karla

Vázquez Sánchez Erick Alejandro

Proyecto Final

Índice

Proyecto final de la asignatura base de datos

1. Objetivo

El alumno analizará una serie de requerimientos y propondrá una solución que atienda los mismos, aplicando los conceptos vistos en el curso.

2. Introducción

En este proyecto se abordará la construcción de una base de datos para llevar a cabo la administración de un restaurante. Su implementación será utilizando el manejador de bases de datos PostgreSQL.

La base de datos desarrollada le permitirá al restaurante administrar la información de sus empleados, almacenar su catálogo de platillos y bebidas y llevar un registro de las órdenes y clientes. Asimismo, podrá visualizar las factura de los clientes y calcular el monto total por la orden y el monto total por cada platillo o bebida contenidos en cada orden.

Para el diseño de la base de datos partimos primero desde el diseño conceptual, en el cual analizamos las necesidades y requerimientos del restaurante para poder plantear el modelo entidad-relación, y debido a que las especificaciones de los requerimientos no eran representadas de manera precisa, optamos por realizar el modelo entidad-relación extendido. Posteriormente, desarrollamos el diseño lógico de la base de datos a través de la representación intermedia del modelo relacional y finalmente, realizamos la representación final del modelo relacional utilizando la herramienta de software ER/Studio. Una vez obtenido el modelo relacional, empezamos a desarrollar el modelo físico de la base de datos, el cual consistió en la creación de tablas y estructuras utilizando PostgreSQL, para ello nos apoyamos de la herramienta pgAdmin, la cual nos proporciona una interfaz gráfica de la base de datos.

La última fase del desarrollo de la base de datos fue la creación de una interfaz gráfica que permitiera realizar consultas e ingresar información a la base de datos. Esta interfaz gráfica fue creada en Python utilizando las librerías Tkinter y Psycopg2 , las cuales proporcionan lo necesario para implementar lo solicitado por el profesor.

3. Descripción del problema

El problema se divide en dos partes:

Parte uno

Consiste en el diseño de una base de datos.
Un restaurante desea digitalizar su forma de operación, para ello se desarrollará un sistema informático que constará de varios módulos. El que corresponde a la implementación de la base de datos deberá atender el siguiente requerimiento:

Se debe almacenar el RFC, número de empleado, nombre, fecha de nacimiento, teléfonos, edad, domicilio, sueldo; de los cocineros su especialidad, de los meseros su horario y de los administrativos su rol, así como una foto de los empleados y considerar que un empleado puede tener varios puestos. Es necesario tener registro de los dependientes de los empleados, su curp, nombre y parentesco. Se debe tener disponible la

información de los platillos y bebidas que el restaurante ofrece, una descripción, nombre, su receta, precio y un indicador de disponibilidad, así como el nombre y descripción de la categoría a la que pertenecen (considerar que un platillo o bebida sólo pertenece a una categoría). Debe tenerse registro del folio de la orden, fecha, la cantidad total a pagar por la orden y registro del mesero que levantó la orden, así como la cantidad de cada platillo/bebida y precio total a pagar por platillo/bebida contenidos en cada orden. Considerar que es posible que los clientes soliciten factura de su consumo, por lo que debe almacenarse su RFC, nombre, domicilio, razón social, email y fecha de nacimiento. Adicional al almacenamiento de información, la base de datos debe atender los siguientes puntos:

- Cada que se agregue un producto a la orden, debe actualizarse los totales(por producto y venta), así como validar que el producto esté disponible.
- Crear al menos, un índice, del tipo que se prefiera y donde se prefiera.Justificar el porqué de la elección en ambos aspectos.
- Dado un número de empleado, mostrar la cantidad de órdenes que ha registrado en el día así como el total que se ha pagado por dichas órdenes. Si no se trata de un mesero, mostrar un mensaje de error.
- Vista que muestre todos los detalles del platillo más vendido.
- Permitir obtener el nombre de aquellos productos que no estén disponibles
- De manera automática se genere una vista que contenga información necesaria para asemejarse a una factura de una orden.
- Dada una fecha, o una fecha de inicio y fecha de fin, regresar el total del número de ventas y el monto total por las ventas en ese periodo de tiempo. Nota: Con producto se hace referencia a los alimentos y bebidas.

Tomar en cuenta las siguientes consideraciones:

- Puede haber distintas soluciones al problema.
- Los requerimientos en listados anteriormente, deberán ser realizados por medio de PgSQL, con los elementos que se consideren adecuados para resolverlos.
- El folio de la orden debe tener un formato similar a ORD-001, prefijo ORD, seguido de un guión y un número secuencial.
- Donde esté presente el atributo domicilio, está compuesto por estado,código postal, colonia, calle y número.
- Donde esté presente el atributo nombre, está compuesto por nombre, apellido paterno y materno.
- El diseño debe satisfacer todos los principios de diseño, los requerimientos anteriores y un buen manejo de información.

Parte dos

Una vez diseñada y lista la base de datos, se debe crear una interfaz gráfica vía app móvil o web, que permita:

1. Consultar la información general de los empleados, incluyendo su fotografía.
2. Ingresar una orden, de hasta 3 productos, los cuales podrán seleccionarse de una lista de opciones, permitir ingresar la cantidad, calcular el costo total de cada artículo y el costo total de toda la orden. Ingresar dicha información en la base de datos, respetando todas las restricciones de integridad

4. Plan de trabajo

Se analizaron los requerimientos para diseñar un Modelo Entidad Relación en la herramienta Diagrams que cumpliera con todos los requisitos solicitados en la descripción del problema. Una vez realizado este modelo, se realizó el mapeo intermedio al Modelo Relacional que después se pasó al mapeo final usando la herramienta ERStudio. Al tener el diagrama del Modelo Relacional, comenzamos a realizar el código SQL para PostgreSQL en el cual creamos la base de datos del restaurante, la creación de cada tabla en nuestro Modelo Relacional junto con sus atributos. Para este punto, tomamos en cuenta cada característica de cada tabla como las relaciones entre sí y además tuvimos en cuenta que había atributos que debían ser calculados mediante otros atributos, y para resolver esto hicimos uso de triggers o disparadores, así como también los usamos, junto con la creación de vistas, para resolver ciertos requerimientos plasmados en la descripción del problema. De igual manera se hizo uso de la herramienta pgAdmin para visualizar de mejor manera nuestra base de datos al momento de ingresar registros e ir probando si nuestra solución iba siendo adecuada. Una vez creada toda la base de datos con su código SQL, comenzamos a crear la interfaz en la que íbamos a mostrar nuestra base de datos.

4.1. Plan de actividades

La implementación de la base de datos constó de varias actividades y para la realización de cada de estas, organizamos el trabajo como se describe a continuación.

	Actividad	Responsable(s)
Diseño conceptual	Modelo entidad-relación	Todos los integrantes del equipo
Diseño lógico	Representación intermedia del MER Representación final del MER Normalización	Todos los integrantes del equipo
Diseño físico	Definición de tablas y estructuras de datos (DDL y DML) Archivos .csv (registros)	Karla Vázquez Pérez y Arely De La Cruz Munguía. Andrea Andrés Urbano
Implementación	Interfaz gráfica	Erik Jonathan Calvillo Martínez Erick Alejandro Vázquez Sánchez

A continuación detallaremos las actividades que realizó cada miembro del equipo:

1. La creación del Modelo Entidad Relación y el Modelo Relacional fue realizada por los 5 integrantes del equipo.
2. La realización del código SQL, lo que corresponde a la creación de las tablas, disparadores, vistas e índices, le correspondió a las integrantes Karla Vázquez Pérez y Arely De La Cruz Munguía.
3. La creación de toda la interfaz y conexión entre esta misma con la base de datos creada le correspondió a los integrantes Erik Jonathan Calvillo Martínez y Erick Alejandro Vázquez Sánchez.
4. La realización de los archivos csv en los cuales se insertaron registros para las tablas de nuestra base de datos fueron realizados por Andrea Andrés Urbano.

5. Diseño de la base de datos

El diseño de la base de datos está compuesto por diferentes fases: diseño conceptual, diseño lógico y diseño físico. A continuación se detalla las decisiones tomadas en cada una de estas fases de diseño.

5.1. Diseño conceptual

5.1.1. Modelo entidad-relación

El diseño conceptual de la base de datos está conformado por la representación de los requerimientos del problema en un modelo entidad-relación. Para el diseño del modelo entidad relación fue necesario analizar a

gran detalle la descripción del problema, de esta forma se pudieron encontrar todas las entidades que participarían dentro de nuestra propuesta de solución así como los atributos correspondientes para cada una de estas. Es importante recalcar que se hizo uso del concepto de MER extendido ya que se considero adecuado para obtener una mejor solución e implementación del problema.

Lo siguiente que se realizo fue el análisis para encontrar de que forma se relacionaban estas entidades, ya que de esta forma se logra entender como es la interacción entre estos en el entorno real. Una vez que se tienen en claro todos estos elementos por separado se implemento el diseño con ayuda de la herramienta "Diagram Software and Flowchart Maker". Esta herramienta tiene los elementos necesarios para lograr implementar de manera gráfica todos los elementos que corresponden al Modelo Entidad Relación.

Dentro del análisis realizado se encontraron las siguientes entidades:

1. EMPLEADO: Esta entidad se considero como un supertipo ya que había diferentes tipos de empleados que compartían ciertas características en común, además de esto se hizo la consideración de utilizar Especialización debido a las características encontradas dentro de los requerimientos. La participación de los subtipos con esta entidad supertipo fue tomada como total y se hizo uso del traslape debido a que estos empleados pueden tener más de un puesto dentro del Restaurante.
   Los atributos de esta entidad son los siguientes: numero de empleado con la restricción de que es la llave primaria, rfc que tendrá una restricción de llave candidata (unique), domicilio que se descompone en código postal, estado, colonia, numero y calle, nombre(s), apellido paterno, apellido materno que puede ser opcional (nulo), fecha de nacimiento, edad que tiene una restricción de tipo check y nos indica que puede ser calculado a partir de la fecha de nacimiento además de una restricción null, foto y por ultimo el atributo teléfono que será considerado como un atributo multivaluado.

2. ADMINISTRADOR: Esta entidad se considero como un subtipo de Empleado que tiene como atributo distintivo el rol que desempeña dentro del Restaurante. Al ser un subtipo hereda los atributos del supertipo Empleado.

3. COCINERO: Esta entidad se considero como un subtipo de Empleado que tiene como atributo distintivo la especialidad que tiene dentro del Restaurante. Al ser un subtipo hereda los atributos del supertipo Empleado.

4. MESERO: Esta entidad se considero como un subtipo de Empleado que tiene como atributo distintivo el horario en el que trabaja en el Restaurante. Al ser un subtipo hereda los atributos del supertipo Empleado.

5. DEPENDIENTE: Esta entidad se considero como una entidad débil ya que no todos los empleados tienen un dependiente y para que esta exista es necesario que haya un empleado trabajando dentro del Restaurante. Los atributos que tiene esta entidad son los siguientes: parentesco con el empleado, CURP que se consideró como un discriminante(se asemeja a la restricción de llave primaria), nombre(s), apellido paterno y apellido materno que es un atributo opcional(restricción null).

6. CLIENTE: Esta entidad tiene los siguientes atributos: RFC que se considero con la restricción de llave primaria, nombre(s), apellido paterno, apellido materno como un atributo opcional (restricción null), domicilio que se descompone en código postal, estado, colonia, numero y calle, fecha de nacimiento, razón social y email.

7. ORDEN: Esta entidad tiene los siguientes atributos: fecha de la orden, precio total por la orden que es un atributo calculado y tiene una restricción null, cantidad total de productos incluidos en la orden que es un atributo calculado y tiene una restricción null, por último el folio de la orden que debe considerar la restricción check en la cual debe llevar el siguiente formato 'ORD-001' en donde la segunda parte debe ser un numero secuencial y la restricción de llave primaria.

8. PRODUCTO: Esta entidad se considero como un supertipo ya que había diferentes productos como platillos y bebidas que compartían características en común, pero en este caso se hizo uso de generalización, con una participación total y con una relación de exclusión. Los atributos de esta entidad son los siguientes: disponibilidad, nombre del producto, cantidad vendida que es un atributo calculado,

descripción del producto, receta, cantidad vendida que es un atributo calculado y por lo tanto tiene la restricción null, precio, cantidad vendida que es calculado y además opcional y finalmente el id del producto que tiene la restricción de llave primaria.

9. PLATILLO: Esta entidad es un subtipo de producto y el atributo que tiene como distintivo es aquel que nos indica si es el platillo del día.

10. BEBIDA: Esta entidad es un subtipo de producto y el atributo que tiene como distintivo es si esta bebida contiene o no alcohol.

11. CATEGORIA: Esta entidad contiene los siguientes atributos: descripción de la categoría, nombre de la categoría y un identificador de la categoría que funcionara como una restricción de llave primaria.

Así mismo, establecimos la siguientes relaciones:

1. EMPLEADO-DEPENDIENTE: Tenemos una relación en la cual la cardinalidad final es de (1,M), por lo que se debe considerar que al pasar al Modelo Relacional deberemos tener la llave primaria del supertipo EMPLEADO como llave foránea en la entidad DEPENDIENTE. El nombre de la relación es "tiene".

2. MESERO-ORDEN: Tenemos una relación en la cual la cardinalidad final es de (1,M), por lo que se debe considerar que al pasar a la representación del Modelo Relacional se deberá tener la llave primaria de la entidad MESERO como llave foránea en la entidad ORDEN. El nombre de la relación es "levanta".

3. CLIENTE-ORDEN: Tenemos una relación en la cual la cardinalidad final es de (1,M), por lo que se debe considerar que al pasar a la representación del Modelo Relacional se deberá tener a la llave primaria de la entidad CLIENTE como llave foránea en la entidad ORDEN. El nombre de la relación es "pide".

4. ORDEN-PRODUCTO:Tenemos una relación en la cual la cardinalidad final es de (M,M), la relación se llama ïncluyez contiene 2 atributos que son: cantidad por producto y precio total por producto el cual será calculado. Dada la cardinalidad se deberá crear una relación extra en la representación del Modelo Relacional con este nombre y con los atributos correspondientes así como las consideraciones de llave primaria que sera compuesta por la llave primaria de ORDEN y por la llave primaria de PRODUCTO.

5. PRODUCTO-CATEGORIA: Tenemos una relación en la cual la cardinalidad final es de (M,1), por lo que se debe considerar que al pasar a la representación del Modelo Relacional se deberá tener a la llave primaria de CATEGORIA como llave foránea en la entidad PRODUCTO. El nombre de la relación es "pertenecer"

A continuación se muestra como es que quedo el diseño del Modelo Entidad Relación (MER) final.

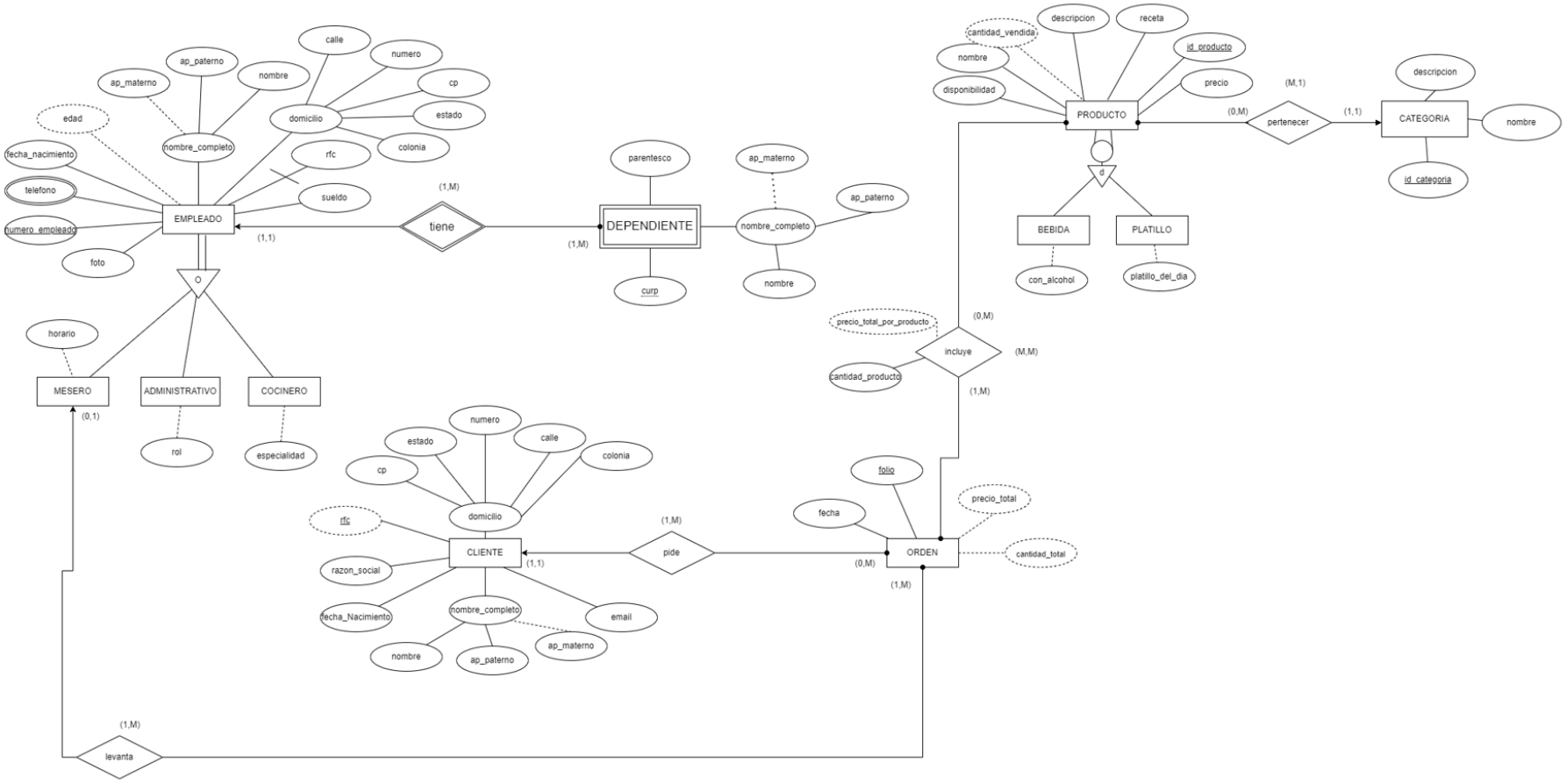


Figura 1: Modelo entidad-relación extendido

5.2. Diseño lógico

5.2.1. Mapeo intermedio del modelo relacional

Una vez teniendo el modelo entidad relación se realizó el mapeo intermedio al modelo relacional para poder identificar mejor las tablas que se deben crear en el mapeo final al modelo relacional.

Se procedió después a la creación del modelo relacional en la herramienta .ER/Studio"basándonos en el mapeo intermedio realizado, tomando en cuenta las restricciones planteadas para cada tabla y atributo, así como también las relaciones entre cada tabla para su adecuada interacción. La herramienta .ER/Studio"mencionada sirve para el diseño de modelado de datos y que en este caso la hemos implementado para realizar el modelo relacional de nuestra base de datos.

Para hacer el mapeo intermedio al modelo relacional de cada entidad y relación se consideró lo siguiente:

1. EMPLEADO: Para esta entidad se consideró la estrategia 4 de mapeo de modelo entidad relación extendido en la cual se crea una sola relación llamada como el supertipo y en la que se incluyen los atributos de sus subtipos, en este caso Mesero, Administrativo y Cocinero, como nulos y tres banderas de tipo bit en las cuales se especifica qué tipo de empleado es. Se ha escogido esta estrategia de mapeo debido a que existe traslape y una relación total, es decir, un empleado puede tener 1 o más roles.

   a) numero_empleado: Este atributo es la llave primaria y es de tipo int.

   b) rfc: Este atributo es de tipo varchar de longitud 13, además de que se indica que es de tipo unique, ya que es llave candidata.

   c) foto: Este atributo es de tipo varchar de longitud 500.

   d) nombre: Este atributo es de tipo varchar de longitud 60.

   e) ap_paterno: Este atributo es de tipo varchar de longitud 60.

   f) ap_materno: Este atributo es de tipo varchar de longitud 60 y además es nulo.

   g) edad: Este atributo es de tipo smallint, de tipo nulo y además calculado.

   h) sueldo: Este atributo es de tipo float.

   i) calle: Este atributo es de tipo varchar de longitud 80.

   j) numero: Este atributo es de tipo smallint.

   k) cp: Este atributo es de tipo int.

   l) estado: Este atributo es de tipo varchar de longitud 60.

   m) colonia: Este atributo es de tipo varchar de longitud 100.

   n) fecha_nacimiento: Este atributo es de tipo date.

   ñ) es_mesero: Este atributo es de tipo boolean.

   o) es_administrativo: Este atributo es de tipo boolean.

   p) es_cocinero: Este atributo es de tipo boolean.

   q) horario: Este atributo es de tipo varchar de longitud 60 y además es nulo.

   r) rol: Este atributo es de tipo varchar de longitud 60 y además es nulo.

   s) especialidad: Este atributo es de tipo varchar de longitud 60 y además es nulo.

2. TELEFONO: Esta tabla se creó debido a que telefono es un atributo multivaluado de la entidad Empleado.

   a) telefono: Este atributo es la llave primaria y es de tipo bigint.

   b) numero_empleado: Este atributo es llave foránea que hace referencia al atributo numero_empleado de la tabla EMPLEADO y es de tipo entero.

3. PRODUCTO: Para esta entidad se consideró la estrategia 3 de mapeo de modelo entidad relación extendido en la cual se crea una sola relación llamada como el supertipo y en la que se incluyen los atributos de sus subtipos, en este caso Platillo y Bebida, como nulos y una bandera de tipo varchar en las cual se especifica qué tipo de producto es. Se ha escogido esta estrategia de mapeo debido a que existe exclusión y una relación total, es decir, un producto sólo puede ser de un tipo.

   a) id_producto: Este atributo es la llave primaria y es de tipo int.

   b) nombre: Este atributo es de tipo varchar de longitud 150.

   c) receta: Este atributo es de tipo varchar de longitud 2000.

   d) precio: Este atributo es de tipo real.

   e) descripcion: Este atributo es de tipo varchar de longitud 400.

   f) disponibilidad: Este atributo es de tipo boolean.

   g) tipo_producto: Este atributo es de tipo varchar de longitud 8.

   h) con_alcohol: Este atributo es de tipo boolean y además es nulo.

   i) platillo_del_dia: Este atributo es de tipo boolean y además es nulo.

   j) cantidad_vendida: Este atributo es de tipo entero, además es nulo y calculado.

   k) id_categoria: Este atributo es llave foránea que hace referencia al atributo id_categoria en la tabla CATEGORIA y además es de tipo int.

4. CLIENTE: Para esta entidad se aplicó una estrategia de mapeo de un modelo entidad relación normal, es decir, no extendido.

   a) rfc: Este atributo es la llave primaria y es de tipo varchar de longitud 13.

   b) nombre: Este atributo es de tipo varchar de longitud 60.

   c) ap_paterno: Este atributo es de tipo varchar de longitud 60.

   d) ap_materno: Este atributo es de tipo varchar de longitud 60 y además es nulo.

   e) fecha_nacimiento: Este atributo es de tipo date.

   f) razon_social: Este atributo es de tipo varchar de longitud 150.

   g) calle: Este atributo es de tipo varchar de longitud 80.

   h) numero: Este atributo es de tipo smallint.

   i) cp: Este atributo es de tipo int.

   j) estado: Este atributo es de tipo varchar de longitud 60.

   k) colonia: Este atributo es de tipo varchar de longitud 100.

   l) email: Este atributo es de tipo varchar de longitud 200.

5. ORDEN: Para esta entidad se aplicó una estrategia de mapeo de un modelo entidad relación normal, es decir, no extendido.

   a) folio: Este atributo es la llave primaria, es de tipo varchar de longitud 20 y además es calculado.

   b) fecha: Este atributo es de tipo date.

   c) precio_total: Este atributo es de tipo real, además es de tipo nulo y calculado.

   d) cantidad_total: Este atributo es de tipo smallint, además es de tipo nulo y calculado.

   e) numero_empleado: Este atributo es llave foránea y hace referencia al atributo numero_empleado en la tabla Empleado, además es de tipo int.

   f) rfc: Este atributo es llave foránea y hace referencia al atributo rfc de la tabla CLIENTE, además es de tipo varchar de longitud 13.

6. CATEGORIA: Para esta entidad se aplicó una estrategia de mapeo de un modelo entidad relación normal, es decir, no extendido.

a) id_categoria: Este atributo es la llave primaria y es de tipo int.

b) nombre: Este atributo es de tipo varchar y de longitud 50.

c) descripcion: Este atributo es de tipo varchar y de longitud 500.

7. DEPENDIENTE:Para esta entidad se aplicó una estrategia de mapeo de un modelo entidad relación normal, es decir, no extendido.

   a) curp: Este atributo es el discriminante y a su vez la llave primaria, es de tipo varchar con una longitud de 18.

   b) nombre: Este atributo es de tipo varchar y de longitud 60.

   c) ap_paterno: Este atributo es de tipo varchar y de longitud 60.

   d) ap_materno: Este atributo es de tipo varchar de longitud 60 y además es nulo.

   e) parentesco: Este atributo es de tipo varchar y de longitud 30.

   f) numero_empleado: Este atributo es llave foránea y hace referencia al atributo numero_empleado de la tabla EMPLEADO, además es de tipo int.

8. INCLUYE: Esta relación surgió como resultado de una relación (M,M) entre la entidad ORDEN y PRODUCTO. Se considero a la llave primaria compuesta por las 2 llaves primarias de las entidades en cuestión que por si solas pasaron como llaves foráneas.

   a) id_producto: Este atributo es llave foránea y a su vez llave primaria, hace referencia al atributo id_producto de la tabla PRODUCTO y es de tipo entero

   b) folio: Este atributo es llave foránea y a su vez llave primaria, hace referencia al atributo folio de la tabla ORDEN y es de tipo varchar de longitud 20.

   c) cantidad_producto: Este atributo es de tipo int y es calculado.

   d) precio_total_por_producto: Este atributo es de tipo float, nulo y es calculado.

En conjunto, el mapeo intermedio del modelo relacional es el siguiente:
EMPLEADO:{numero_empleado int(PK), rfc varchar(13)(U), nombre varchar(60), ap_paterno varchar(60), ap_materno varchar(60)(N), edad smallint(N)(C), sueldo float, calle varchar(80), numero smallint, cp int, estado varchar(60), colonia varchar(100), fecha_nacimiento date,es_mesero boolean, es_administrativo boolean, es_cocinero boolean, horario varchar(60)(N), rol varchar (60)(N), especialidad varchar(60)(N), foto varchar(500)}

TELEFONO:{telefono bigint(PK), numero_empleado int(FK)}

PRODUCTO: {id_producto int(PK), nombre varchar(150), receta varchar(2000), precio float, descripcion varchar(400), disponibilidad boolean, tipo_producto varchar(8), con_alcohol boolean(N), platillo_del_dia boolean(N), cantidad_vendida int(N)(C), id_categoria int(FK)(N)}

CLIENTE:{rfc varchar(13)(PK), nombre varchar(60), ap_paterno varchar(60), ap_materno varchar(60)(N), fecha_nacimiento date, razon_social varchar(150), cp int, estado varchar(60), numero smallint, calle varchar (80), colonia varchar (100), email varchar(200)}

ORDEN: {folio varchar(20)(C)(PK), fecha date, precio_total real(N)(C), cantidad_total smallint(N)(C), numero_empleado int(FK)(N), rfc varchar(13)(FK)(N)}

CATEGORIA: {id_categoria int(PK), nombre varchar(50), descripcion varchar(500)}

DEPENDIENTE:{curp varchar(18)(D)(PK), nombre varchar (60), ap_Paterno varchar(60), ap_materno varchar(60)(N), parentesco varchar(30), numero_empleado int(FK)}

INCLUYE:{[id_producto int(FK), folio varchar(20)(FK)](PK), cantidad_producto int, precio_total_por_producto

float(N)(C)}

5.2.2. Normalización

Antes de pasar a la representación final es importante verificar que las relaciones (tablas) obtenidas a partir del modelo relacional estén normalizadas, así que a continuación se analizará a cada una de las tablas para corroborar que están en 3FN o en su defecto realizar la normalización correspondiente.

1. Relación EMPLEADO
   La relación empleado esta definida como se muestra a continuación.

   empleado (numero_empleado, foto, rfc, nombre, ap_paterno, ap_materno, edad, sueldo, calle, numero, cp, estado, colonia, fecha_nacimiento, es_mesero, es_administrativo, es_cocinero, horario, rol, especialidad)

   Dado que el número de columnas de la relación es demasiado grande para ser representado en una sola tabla sin salirse del documento, se partirá, sin embargo, es la representación de una sola tabla.

numero_empleado	foto	rfc	nombre	ap_paterno	ap_materno	edad	sueldo	calle	numero	cp

estado	colonia	fecha_nacimiento	es_mesero	es_administrativo	es_cocinero	horario	rol	especialidad

   a) **Llave primaria**

   PK:{numero_empleado}

   b) **Dependencias funcionales**

   {numero_empleado}→{foto, rfc, nombre, ap_paterno, ap_materno, edad, sueldo, calle, numero, cp, estado, colonia, fecha_nacimiento, es_mesero, es_administrativo, es_cocinero, horario, rol, especialidad}

   - **Dependencias parciales**: La relación no tiene dependencias parciales.
   - **Dependencias transitivas**: La relación no tiene dependencias transitivas.

   c) **Formas normales**

   1) **Primera forma normal (1FN)**
      Una relación está en 1FN si tiene dominios simples, es decir, no hay grupos de repetición y todos sus valores son atómicos, es decir, no hay atributos multivaluados. Analizando en la relación:
      - Atributos multivaluados: La relación no contiene atributos multivaluados.
      - Grupos de repetición: No hay grupos de repetición.

      Por lo tanto, la relación se encuentra normalizada en 1FN.

   2) **Segunda forma normal(2FN)**
      Una relación está en 2FN si y sólo si está en 1FN y no tiene dependencias parciales. Analizando en la relación:
      - Llave primaria simple o compuesta: La llave primaria es simple.
      - Dependencias funcionales parciales: La relación no contiene dependencias parciales.

Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**
Una relación está en 3FN si y sólo si está en 2FN y no tiene dependencias transitivas. Analizando en la relación:
- Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

Por lo tanto, la relación se encuentra normalizada en 3FN.

2. Relación TELEFONO
La relación empleado esta definida como se muestra a continuación.

telefono	numero_empleado

a) **Llave primaria**

PK:{telefono}

b) **Dependencias funcionales**

{telefono}→{numero_empelado}

- **Dependencias parciales**: La relación no tiene dependencias parciales.
- **Dependencias transitivas**: La relación no tiene dependencias transitivas.

c) **Formas normales**

1) **Primera forma normal (1FN)**
Analizando en la relación las condiciones para que se encuentre en 1FN:
- Atributos multivaluados: La relación no contiene atributos multivaluados.
- Grupos de repetición: No hay grupos de repetición.

Por lo tanto, la relación se encuentra normalizada en 1FN.

2) **Segunda forma normal(2FN)**
Analizando en la relación las condiciones para que se encuentre en 2FN:
- Llave primaria simple o compuesta: La llave primaria es simple.
- Dependencias funcionales parciales: La relación no contiene dependencias parciales.

Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**
Analizando en la relación las condiciones para que se encuentre en 3FN:
- Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

Por lo tanto, la relación se encuentra normalizada en 3FN.

3. Relación PRODUCTO
La relación empleado esta definida como se muestra a continuación.

id_producto	nombre	receta	precio	descripcion	disponibilidad	tipo_producto	id_categoria

a) **Llave primaria**

PK:{id_producto}

b) **Dependencias funcionales**

{id_producto}→{nombre, receta, precio, descripcion, disponibilidad, tipo_producto, id_categoria}

- **Dependencias parciales**: La relación no tiene dependencias parciales.
- **Dependencias transitivas**: La relación no tiene dependencias transitivas.

c) **Formas normales**

1) **Primera forma normal (1FN)**
Analizando en la relación las condiciones para que se encuentre en 1FN:
   - Atributos multivaluados: La relación no contiene atributos multivaluados.
   - Grupos de repetición: No hay grupos de repetición.

   Por lo tanto, la relación se encuentra normalizada en 1FN.

2) **Segunda forma normal(2FN)**
Analizando en la relación las condiciones para que se encuentre en 2FN:
   - Llave primaria simple o compuesta: La llave primaria es simple.
   - Dependencias funcionales parciales: La relación no contiene dependencias parciales.

   Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**
Analizando en la relación las condiciones para que se encuentre en 3FN:
   - Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

   Por lo tanto, la relación se encuentra normalizada en 3FN.

4. Relación CLIENTE
La relación empleado esta definida como se muestra a continuación.

cliente(rfc. email, nombre, ap_paterno, ap_materno, fecha_nacimiento, razon_social, cp, estado, numero, calle, colonia)

Dado que el número de columnas de la relación es demasiado grande para ser representado en una sola tabla sin salirse del documento, se partirá, sin embargo, es la representación de una sola tabla.

rfc	email	nombre	ap_paterno	ap_materno	fecha_nacimiento	razon_social	cp	estado

numero	calle	colonia

a) **Llave primaria**

PK:{rfc}

b) **Dependencias funcionales**

{rfc}→{email, nombre, ap_paterno, ap_materno, fecha_nacimiento, razon_social, cp, estado, numero, calle, colonia}

- **Dependencias parciales**: La relación no tiene dependencias parciales.

- **Dependencias transitivas**: La relación no tiene dependencias transitivas.

c) **Formas normales**

1) **Primera forma normal (1FN)**
Analizando en la relación las condiciones para que se encuentre en 1FN:
   - Atributos multivaluados: La relación no contiene atributos multivaluados.
   - Grupos de repetición: No hay grupos de repetición.

   Por lo tanto, la relación se encuentra normalizada en 1FN.

2) **Segunda forma normal(2FN)**
Analizando en la relación las condiciones para que se encuentre en 2FN:
   - Llave primaria simple o compuesta: La llave primaria es simple.
   - Dependencias funcionales parciales: La relación no contiene dependencias parciales.

   Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**
Analizando en la relación las condiciones para que se encuentre en 3FN:
   - Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

   Por lo tanto, la relación se encuentra normalizada en 3FN.

5. Relación ORDEN
La relación empleado esta definida como se muestra a continuación.

folio	fecha	precio_total	cantidad_total	numero_empleado	rfc

a) **Llave primaria**

PK:{folio}

b) **Dependencias funcionales**

{folio}→{fecha, precio_total, cantidad_total, numero_empleado, rfc}

- **Dependencias parciales**: La relación no tiene dependencias parciales.
- **Dependencias transitivas**: La relación no tiene dependencias transitivas.

c) **Formas normales**

1) **Primera forma normal (1FN)**
Analizando en la relación las condiciones para que se encuentre en 1FN:
   - Atributos multivaluados: La relación no contiene atributos multivaluados.
   - Grupos de repetición: No hay grupos de repetición.

   Por lo tanto, la relación se encuentra normalizada en 1FN.

2) **Segunda forma normal(2FN)**
Analizando en la relación las condiciones para que se encuentre en 2FN:
   - Llave primaria simple o compuesta: La llave primaria es simple.
   - Dependencias funcionales parciales: La relación no contiene dependencias parciales.

Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**
Analizando en la relación las condiciones para que se encuentre en 3FN:
- Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

Por lo tanto, la relación se encuentra normalizada en 3FN.

6. Relación CATEGORIA
La relación empleado esta definida como se muestra a continuación.

id_categoria	nombre	descripcion

a) **Llave primaria**

PK:{id_categoria}

b) **Dependencias funcionales**

{id_categoria}→{nombre, descripcion}

- **Dependencias parciales**: La relación no tiene dependencias parciales.
- **Dependencias transitivas**: La relación no tiene dependencias transitivas.

c) **Formas normales**

1) **Primera forma normal (1FN)**
Analizando en la relación las condiciones para que se encuentre en 1FN:
- Atributos multivaluados: La relación no contiene atributos multivaluados.
- Grupos de repetición: No hay grupos de repetición.

Por lo tanto, la relación se encuentra normalizada en 1FN.

2) **Segunda forma normal(2FN)**
Analizando en la relación las condiciones para que se encuentre en 2FN:
- Llave primaria simple o compuesta: La llave primaria es simple.
- Dependencias funcionales parciales: La relación no contiene dependencias parciales.

Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**
Analizando en la relación las condiciones para que se encuentre en 3FN:
- Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

Por lo tanto, la relación se encuentra normalizada en 3FN.

7. Relación DEPENDIENTE
La relación empleado esta definida como se muestra a continuación.

curp	nombre	ap_paterno	ap_materno	parentesco	numero_empleado

a) **Llave primaria**

PK:{curp}

b) **Dependencias funcionales**

{curp}→{nombre, ap_paterno, ap_materno, parentesco, numero_empleado}

- **Dependencias parciales**: La relación no tiene dependencias parciales.
- **Dependencias transitivas**: La relación no tiene dependencias transitivas.

c) **Formas normales**

1) **Primera forma normal (1FN)**
Analizando en la relación las condiciones para que se encuentre en 1FN:
   - Atributos multivaluados: La relación no contiene atributos multivaluados.
   - Grupos de repetición: No hay grupos de repetición.

   Por lo tanto, la relación se encuentra normalizada en 1FN.

2) **Segunda forma normal(2FN)**
Analizando en la relación las condiciones para que se encuentre en 2FN:
   - Llave primaria simple o compuesta: La llave primaria es simple.
   - Dependencias funcionales parciales: La relación no contiene dependencias parciales.

   Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**
Analizando en la relación las condiciones para que se encuentre en 3FN:
   - Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

   Por lo tanto, la relación se encuentra normalizada en 3FN.

8. Relación INCLUYE
La relación empleado esta definida como se muestra a continuación.

folio	id_producto	cantidad_producto	precio_total_por_producto

a) **Llave primaria**

PK:{folio, id_producto}

b) **Dependencias funcionales**

{folio, id_producto}→{cantidad_producto, precio_total_por_producto}

- **Dependencias parciales**: La relación no tiene dependencias parciales.
- **Dependencias transitivas**: La relación no tiene dependencias transitivas.

c) **Formas normales**

1) **Primera forma normal (1FN)**

   Una relación está en 1FN si tiene dominios simples, es decir, no hay grupos de repetición y todos sus valores son atómicos, es decir, no hay atributos multivaluados. Analizando en la relación:
   - Atributos multivaluados: La relación no contiene atributos multivaluados.

- Grupos de repetición: No hay grupos de repetición.

Por lo tanto, la relación se encuentra normalizada en 1FN.

2) **Segunda forma normal(2FN)**

Una relación está en 2FN si y sólo si está en 1FN y no tiene dependencias parciales. Analizando en la relación:

- Llave primaria simple o compuesta: La llave primaria es simple.
- Dependencias funcionales parciales: La relación no contiene dependencias parciales.

Dado que la llave primaria es simple, no hay dependencias parciales y por lo tanto, la relación se encuentra normalizada en 2FN.

3) **Tercera forma normal (3FN)**

Una relación está en 3FN si y sólo si está en 2FN y no tiene dependencias transitivas. Analizando en la relación:

- Dependencias funcionales transitivas: La relación no contiene dependencias transitivas.

Por lo tanto, la relación se encuentra normalizada en 3FN.

5.2.3. Mapeo final del modelo relacional

Una vez normalizadas cada una de las tablas obtenidas en el mapeo intermedio del modelo relacional, procedemos a realizar la representación final del modelo relacional. La representación de las tablas y de las relaciones se hizo por medio de la notación Crow's Foot considerando y respetando cada una de las cardinalidades que ya se tenían establecidas en el Modelo Entidad Relación. Finalmente este fue el resultado final después de su implementación en ER studio.

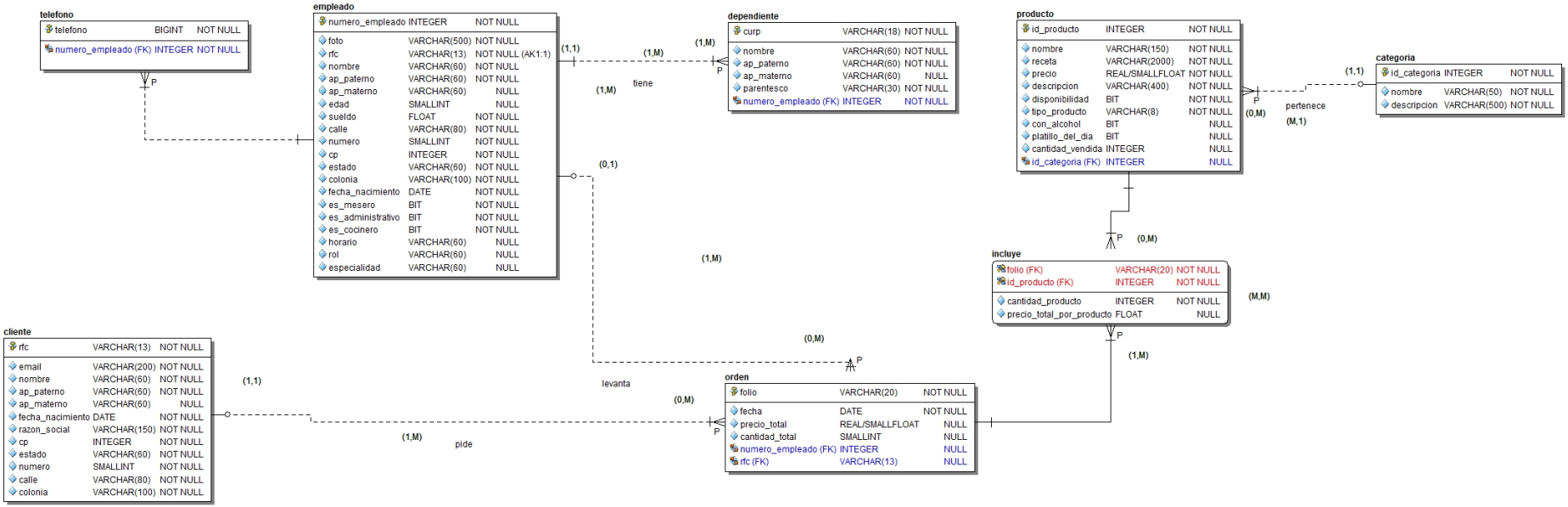


Figura 2: Modelo relacional

6. Implementación

6.1. Diseño físico

El diseño físico corresponde al desarrollo del código SQL para la creación de la base de datos. A continuación de detallan las características y decisiones técnicas que se tuvieron que tomar para crear cada una de tablas, estructuras y funcionalidades de la base de datos.

6.1.1. Especificaciones de la base de datos

Se analizaron las características que debe tener la base de datos para respetar al usuario que será dueño de esta base, así como el tipo de codificación que manejara para su conexión y manejo de datos, no se especifico si esta debía tener un nombre en especifico así que se decidió nombrarla únicamente como RESTAURANTE". Únicamente se hizo uso de la siguiente sentencia para su creación:

CREATE DATABASE RESTAURANTE;
En este caso no establecimos un limite de conexiones y el "dueño" de la base de datos por default fue postgres, ya que desde ahí se creo la base de datos. Estas son las únicas características de la creación de la base de datos, dado que no se especifico alguna otra en los requerimientos.

6.1.2. Creación de tablas

Después de haber creado la base de datos, se decidió hacer el código SQL necesario para la creación de las diversas tablas(relaciones) que forman parte de nuestra propuesta de solución. Fue importante considerar los siguientes puntos para la creación de las tablas:

1. Orden de inserción de las tablas debido a las restricciones de llave foránea que algunas de estas poseen.
2. Restricción NOT NULL/NULL implementada a nivel columna en los atributos.
3. Restricciones CHECK, PRIMARY KEY y FOREIGN KEY implementadas a nivel tabla para poder colocarles un nombre que podamos ser capaces de reconocer y que se puedan utilizar para futuras referencias.
4. Restricciones aplicadas a las llaves foráneas para saber de que manera se va a manejar el proceso de borrado y actualización para respetar la integridad de los datos.

Tabla empleado

Después de tomar en cuenta los puntos anteriores se consideró ingresar la primer tabla que es "EMPLEADO" ya que su llave primaria se propaga como llave foránea en las tablas "DEPENDIENTE" y "ORDEN", por lo que podemos decir que esta es una tabla padre. Se implementó con la siguiente sintaxis:

```
CREATE TABLE empleado(  numero_empleado integer NOT NULL,
        foto character varying(500) NOT NULL,
        rfc character varying(13) NOT NULL,
        nombre character varying(60) NOT NULL,
        ap_paterno character varying(60) NOT NULL,
        ap_materno character varying(60) NOT NULL,
        sueldo float NOT NULL,
        calle character varying(80) NOT NULL,
        numero smallint NOT NULL,
        cp integer NOT NULL,
        estado character varying(60) NOT NULL,
        colonia character varying(100) NOT NULL,
        fecha_nacimiento date NOT NULL,
```

```
    edad integer NULL,
    es_mesero boolean NOT NULL,
    es_administrativo boolean NOT NULL,
    es_cocinero boolean NOT NULL,
    horario character varying(60) NULL,
    rol character varying(60) NULL,
    especialidad character varying(60) NULL,
    CONSTRAINT pk_empleado PRIMARY KEY (numero_empleado),
    CONSTRAINT ak_rfc UNIQUE (rfc)
);
```

Como se puede observar se hizo uso de dos CONSTRAINT a nivel tabla que fueron PRIMARY KEY en la columna numero_empleado y la restricción UNIQUE en la columna rfc ya que esta es una clave candidata.

Tabla telefono

Esta tabla fue la segunda en crearse y surgió debido a que el TELEFONO es un atributo multivaluado, por lo que necesitaba la llave primaria de la tabla empleado para que esta se propague como llave foránea. el código para su creación es el siguiente:

```
CREATE TABLE telefono(  telefono bigint NOT NULL,
    numero_empleado integer NOT NULL,
    CONSTRAINT pk_telefono PRIMARY KEY (telefono),
    CONSTRAINT fk_telefono_empleado FOREIGN KEY (numero_empleado) REFERENCES empleado (numero_empleado) ON UPDATE CASCADE ON DELETE CASCADE
);
```

Esta tabla tiene una restricción de PRIMARY KEY que es la columna telefono y este constraint se definió a nivel de tabla.
Además como se menciono en un principio tiene la restriccion de FOREIGN KEY que es la columna numero_empleado y que posee para la actualización de información la restricción 'ON UPDATE CASCADE', ya que si se actualiza la información del empleado se desea que también se actualiza en la tabla teléfono para no tener que hacer movimientos extras, además se agrego para el borrado de información la restricción 'ON DELETE CASCADE' ya que si se borra la información de un empleado no nos interesa tener su(s) numero(s) de contacto almacenados.

Tabla dependiente

La tabla "DEPENDIENTE" fue la tercer tabla que se decidió insertar debido a que esta tiene como restricción de llave foránea a la llave primaria de la tabla "EMPLEADO", el código para su creación fue el siguiente:

```
CREATE TABLE dependiente(
    curp character varying(18) NOT NULL,
    nombre character varying(60) NOT NULL,
    ap_paterno character varyinc(60) NOT NULL,
    ap_materno character varying(60) NULL,
    parentesco character varying(30) NOT NULL,
    numero_empleado integer NOT NULL,
    CONSTRAINT pk_dependiente PRIMARY KEY (curp),
    CONSTRAINT fk_dependiente_empleado FOREIGN KEY (numero_empleado) REFERENCES empleado (numero_empleado) ON UPDATE CASCADE ON DELETE CASCADE
);
```

Como podemos observar esta tabla tiene la restricción NOT NULL a nivel columna en casi todos sus atributos excepto en el apellido materno ya que este puede ser opcional, además se manejan 2 restricciones a

nivel tabla que son las siguientes:

1. El constraint con la restricción PRIMARY KEY en la columna curp.
2. El constraint con la restricción FOREIGN KEY en la columna numero_empleado en donde se aplico para la actualización de información la restricción 'ON UPDATE CASCADE' ya que al momento de nosotros actualizamos información en la tabla empleado queremos que también se haga en la tabla dependiente para no perder ninguna referencia. También se implemento la restricción 'ON DELETE CASCADE' ya que si nosotros borramos la información de algún empleado no nos parece relevante tener la información de un dependiente que ya no tiene asociado un empleado activo.

Tabla cliente

La cuarta tabla que se creo fue la del cliente ya que su llave primaria se propagaría como llave foránea a la tabla orden (esto considerando que el usuario en cuestión desee tener una factura). El código para su creación es el siguiente:

```
CREATE TABLE cliente(
        rfc character varying(13) NOT NULL,
        email character varying(200) NOT NULL,
        nombre character varying(60) NOT NULL,
        ap_paterno character varying(60) NOT NULL,
        ap_materno character varying(60) NOT NULL,
        fecha_nacimiento date NOT NULL,
        razon_social character varying(150) NOT NULL,
        cp integer NOT NULL,
        estado character varying(60) NOT NULL,
        numero smallint NOT NULL,
        calle character varying(80) NOT NULL,
        colonia character varying(100) NOT NULL,
        CONSTRAINT pk_cliente PRIMARY KEY(rfc)
    );
```

Para esta tabla únicamente se aplicaron las restricciones NOT NULL y NULL en las columnas y la restricción PRIMARY KEY a nivel tabla en la columna rfc.

Tabla orden

La quinta tabla en implementarse fue la tabla "ORDEN" ya que la llave primaria de esta se propaga como llave foránea y a la vez primaria de una relación que surgió de una relación (M,M) con la tabla "PRODUCTO", además ya se puede crear esta tabla por que ya se tiene a la tabla empleado y a la tabla cliente quieres propagan sus llaves primarias como llaves foráneas a esta tabla. El código para su creación es el siguiente:

```
CREATE TABLE orden(
        folio character varying(20) NOT NULL,
        fecha date NOT NULL,
        precio_total float NULL,
        cantidad_total smallint NULL,
        numero_empleado integer NULL,
        rfc character varying(13) NULL,
        CONSTRAINT pk_orden PRIMARY KEY(folio),
        CONSTRAINT fk_orden_cliente FOREIGN KEY (rfc) REFERENCES cliente(rfc) ON UPDATE
CASCADE ON DELETE SET NULL,
        CONSTRAINT fk_orden_empleado FOREIGN KEY (numero_empleado) REFERENCES empleado
(numero_empleado) ON UPDATE CASCADE ON DELETE SET NULL
    );
```

En esta tabla se hizo uso de la restricción NOT NULL y NULL a nivel columna en los atributos, además se hace uso de la restricción PRIMARY KEY en la columna 'folio'. Se tienen 2 constraint de llave foránea con las siguientes características:

1. Constraint de FOREIGN KEY del atributo numero_empleado ya que este atributo es la llave primaria de la tabla 'EMPLEADO' con quien la orden tiene una relación con el subtipo 'MESERO'. Se decidió aplicar para la actualización de información la restricción 'ON UPDATE CASCADE' ya que queremos que la información del empleado se actualice tanto en su tabla como en la tabla orden para no tener problemas de perder referencias por alguna actualización que sea requerida de este, y para el caso del borrado de información se utilizo la restricción 'ON DELETE SET NULL', esto con la finalidad de que si algún empleado es borrado no se pierda la información que se tiene de la orden, si no solo se coloque un valor nulo en la columna que nos permita tener la relación de estas tablas y que así se conserven los datos de la orden para futuras consultas de los ingresos que se tuvieron con las ordenes realizadas.

2. Constrain de FOREIGN KEY del atributo rfc ya que este atributo es la llave primaria de la tabla 'CLIENTE' con quien la orden tiene una relación. Se decidió aplicar para actualizar la información la restricción 'ON UPDATE CASCADE' ya que queremos que la información del cliente se actualice tanto en su tabla como en la tabla orden para no tener problemas de perder referencias por alguna actualización que sea requerida de este, y para el caso del borrado de información se utilizo la restricción 'ON DELETE SET NULL', esto con la finalidad de que si algún cliente es borrado no se pierda la información que se tiene de la orden que este realizo si no que solo se coloque un valor nulo en la columna que nos permita tener la relación de estas tablas y que así se conserven los datos de la orden para futuras consultas de los ingresos que se tuvieron con las ordenes realizadas.

Tabla categoria

Esta tabla fue la sexta tabla en ser creada ya que su llave primaria se propaga como llave foránea a la tabla "PRODUCTO", el código para su creación es el siguiente:

```
CREATE TABLE categoria(
        id_categoria integer NOT NULL,
        nombre character varying(50) NOT NULL,
        descripcion character varying(500) NOT NULL,
        CONSTRAINT pk_categoria PRIMARY KEY (id_categoria)
    );
```

Para esta tabla solo se hizo uso de las restricciones NOT NULL a nivel columna y del constraint con la restricción PRIMARY KEY aplicada en la columna id_categoría.

Tabla producto

Esta fue la séptima tabla que se creo ya que su llave primaria de propaga como foránea a la tabla INCLUYE, además posee una restricción de FOREIGN KEY que hace referencia a la tabla CATEGORIA. El codigo para su creación es el siguiente:

```
CREATE TABLE producto(
        id_producto character varying(40) NOT NULL,
        nombre character varying (150) NOT NULL,
        receta character varying(2000) NOT NULL,
        precio float NOT NULL,
        descripcion character varying(100) NOT NULL,
        disponibilidad boolean NOT NULL,
        tipo_producto character varying(8) NOT NULL,
        con_alcohol boolean NULL,
        platillo_del_dia boolean NULL,
        id_categoria integer NULL,
```

cantidad_vendida integer NULL,
CONSTRAINT ck_precio CHECK (precio>0),
CONSTRAINT pk_producto PRIMARY KEY (id_producto),
CONSTRAINT fk_producto_categoria FOREIGN KEY (id_categoria) REFERENCES categoria (id_categoria) ON UPDATE CASCADE ON DELETE SET NULL
);

En esta tabla se hizo uso de la restriccion NULL Y NOT NULL a nivel columna, y para el caso de las llaves primaria y foránea se plantearon de la siguiente forma

1. El constraint de la restricción PRIMARY KEY se aplicó a la columna 'id_producto'
2. El constraint de CHECK fue aplicado a la columna precio para que este siempre fuera mayor a 0.
3. El constraint de la restriccion FOREIGN KEY se aplico sobre la columna id_categoria que hace referencia a la tabla categoria, se eligió la restricción para actualizar como 'ON UPDATE CASCADE' ya que si se hace alguna modificación en la tabla categoria queremos que esta actualización también afecte a la tabla producto y para el borrado la restricción 'ON DELETE SET NULL' se eligió hacer nulo el borrado ya que si se borra la categoria no se desea perder la información del producto ya que este también estará presente en la relación incluye y orden y estas ya no tendrían la información que se necesita para realizar diversas consultas en diferentes periodos de tiempo.

Tabla incluye

Finalmente se agrego la ultima tabla (octava) que es la resultante de una relación (M,M) entre PRODUCTO y ORDEN, el código para su creación es el siguiente:

CREATE TABLE incluye(
folio character varying(20) NOT NULL,
id_producto integer NOT NULL,
cantidad_producto integer NOT NULL,
precio_total_por_producto float NULL,
CONSTRAINT ck_cantidad_producto CHECK (cantidad_producto>0),
CONSTRAINT pk_incluye PRIMARY KEY (folio, id_producto),
CONSTRAINT fk_incluye_orden FOREIGN KEY (folio) REFERENCES orden ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT fk_incluye_producto FOREIGN KEY (id_producto) REFERENCES producto (id_producto) ON UPDATE CASCADE ON DELETE RESTRICT
);

Aquí tenemos los siguientes constraints:

1. Tenemos el constraint de la restricción PRIMARY KEY que en este caso es una llave primaria compuesta por folio y id_producto.
2. Se hace un constraint de un CHECK para verificar que en la columna cantidad_producto el valor siempre sea mayor a 0.
3. Dadas las caracteristicas de la tabla incluye las llaves primarias tambien se propagaron como foráneas con las siguientes características:
   a) El constraint de la FOREIGN KEY folio se implemento para la actualización de información con la restricción 'ON UPDATE CASCADE' ya que si se actualiza información de la orden también queremos que se actualice en la tabla incluye para tener consistencia en los datos almacenados, y para el borrado se implemento la restricción 'ON DELETE CASCADE' ya que si la orden se cancela y se borra no deseamos almacenar información de algo que no ocurrio realmente.

b) El constraint de la FOREIGN KEY id_producto se implemento para la actualización de información con la restricción 'ON UPDATE CASCADE' ya que si se actualiza información de la orden también queremos que se actualice en la tabla incluye para tener consistencia en la información almacenada y en el caso del borrado se uso la restricción 'ON DELETE RESTRICT' ya que si queremos borrar un producto primero debe ser desde la orden y después en la tabla producto, ya que de esta manera garantizamos que no se pierda información importante en caso de un mal movimiento.

6.1.3. Secuenciadores

Secuenciador implementado:

1. folio:Este secuenciador se hizo con la intención de hacerlo parte del folio de la orden ya que en los requerimientos iniciales se nos indico que este debía tener un numero secuencial asociado a un formato especifico. Este secuenciador se utiliza en una función que concatena ambos valores. Este se definió de la siguiente forma: su valor mínimo es 1, empieza en 1, va aumentando de 1 en 1, su valor máximo es de 99999 y es un secuenciador no cíclico para evitar tener folios repetidos.

6.1.4. Índices

Índice implementado:

1. El índice creado fue de tipo NON CLUSTERED implementado sobre la columna "nombre" en la relación PRODUCTO ya que esta columna será varias accedida para que así el usuario pueda elegir el de su preferencia y al implementarlo ahí nos ayudara a disminuir los tiempos de ejecución y será más rápido.

6.1.5. Disparadores

Disparadores implementados:

1. edad_trigger: Este disparador extrae la parte de una fecha que nosotros le indiquemos, en este caso el año. La fecha de la que se extraerá será la del resultado de usar la función age() que nos calcula la edad dándole la fecha de ahora y la fecha de nacimiento del empleado que está almacenada en fecha_nacimiento en la tabla Empleado. Este disparador le almacena el resultado al atributo edad de la tabla Empleado antes de insertar o actualizar en la tabla empleado en cualquier renglón.

2. cantidad_total_trigger: Este disparadores se va a implementar después de insertar, actualizar o borrar en la tabla Incluye para cada renglón. Lo que hace es actualizar el valor cantidad_total en la tabla Orden con el resultado de una consulta compuesta de subconsultas. Dentro de cláusulas IF e IFELSE en la que entrara a cada uno dependiendo si se va a insertar, actualizar o borrar de la tabla Incluye.
   La primera subconsulta (la subconsulta de una segunda subconsulta) nos devuelve una tabla agrupada por el folio de la tabla incluye y que también tiene una segunda columna que nos da la suma de cantidad_producto de esos folios. Este resultado es la tabla de donde la segunda subconsulta, que es la subconsulta de la consulta principal, va a obtener sus datos. Esta segunda subconsulta nos va a devolver cantidad_producto en donde el folio es igual al folio que están ingresando a la tabla Incluye. Y finalmente este resultado es la tabla de donde la consulta principal va a obtener su resultado. Esta consulta principal nos devuelve cantidad_producto donde el folio en Orden es igual al folio que se está ingresando en la tabla Incluye. Así es como logramos obtener la suma de todos los cantidad_producto de cada orden y se lo asignamos a la tabla Orden como su cantidad_total. En otras palabras, en este trigger calculamos la cantidad total de todos los productos que ha comprado el cliente en una orden.
   Si entra al IF y al IFELSE de cuando va a insertar o actualizar de la tabla Incluye se hace referencia a datos de los atributos a ingresar con un NEW. Si entra al IFELSE de cuando se va a borrar de la tabla Incluye, se hace referencia a los datos de los atributos a borrar con un OLD.

3. precio_total_por_producto_trigger: Este disparador se implementa antes de insertar o actualizar en la tabla Incluye para cada renglón. Lo que hace es almacenar en el atributo precio_total_por_producto en la tabla Incluye el resultado de la multiplicación del valor que se le está ingresando a cantidad_producto

por el resultado de una consulta que te devuelve el precio de la tabla producto donde el id_producto de esta misma tabla es igual al id_producto que se le está ingresando a la tabla Incluye. Así es como calculamos el precio total por producto de una orden.

4. precio_total_trigger: Lo que hace es actualizar el valor precio_total en la tabla Orden con el resultado de una consulta compuesta de subconsultas. Dentro de cláusulas IF e IFELSE en la que entrara a cada uno dependiendo si se va a insertar, actualizar o borrar de la tabla Incluye.
   La primera subconsulta (la subconsulta de otra subconsulta) nos devuelve una tabla agrupada por el folio de la tabla Incluye y que también tiene una segunda columna que nos da la suma de precio_total_por_producto de esos folios. Este resultado es la tabla de donde la segunda subconsulta, que es la subconsulta de la consulta principal, va a obtener sus datos. Esta segunda subconsulta nos va a devolver precio_total_por_producto en donde el folio es igual al folio que están ingresando a la tabla Incluye. Y finalmente este resultado es la tabla de donde la consulta principal va a obtener su resultado. Esta consulta principal nos devuelve precio_total_por_producto donde el folio en Orden es igual al folio que se está ingresando en la tabla Incluye. Así es como logramos obtener la suma de todos los precio_total_por_producto de cada orden y se lo asignamos a la tabla Orden como su precio_total. En otras palabras, en este trigger calculamos el precio total de todos los productos que ha comprado el cliente en una orden, es decir, el precio total de la orden. Si entra al IF y al IFELSE de cuando va a insertar o actualizar de la tabla Incluye se hace referencia a datos de los atributos a ingresar con un NEW. Si entra al IFELSE de cuando se va a borrar de la tabla Incluye, se hace referencia a los datos de los atributos a borrar con un OLD.

5. disponibilidad_producto_trigger: Este disparador se implementa antes de insertar o actualizar en la tabla Incluye para cada renglón. Lo que hace es que dentro de una cláusula IF que va a comparar un FALSE con el resultado de una consulta. Esta consulta nos devuelve lo que hay en disponibilidad de la tabla producto donde el id_producto de esta tabla es igual al id_producto que están intentando ingresar. Por lo que si lo que devuelve esta consulta es FALSE, entra a este if y marca un error de que el Producto no está disponible por el momento. Después tenemos un ELSEIF en el que va a entrar si el id_producto que devuelve una subconsulta no se encuentra en la tabla Producto, es decir, si el id_producto que se quiere ingresar a la tabla Incluye no se encuentra en la tabla Producto, devuelve un error que dice que el producto no existe en el menú. Así es como hacemos que no se pueda ingresar un producto que no está disponible o que no existe a una orden.

6. cantidad_vendida_trigger: Lo que hace es actualizar el valor cantidad_vendida en la tabla Producto con el resultado de una consulta compuesta de subconsultas. Dentro de cláusulas IF e IFELSE en la que entrara a cada uno dependiendo si se va a insertar, actualizar o borrar de la tabla Incluye.
   La primera subconsulta (la subconsulta de otra subconsulta) nos devuelve una tabla agrupada por el id_producto de la tabla Incluye y que también tiene una segunda columna que nos da la suma de cantidad_producto de esos id_producto. Este resultado es la tabla de donde la segunda subconsulta, que es la subconsulta de la consulta principal, va a obtener sus datos. Esta segunda subconsulta nos va a devolver cantidad_producto en donde el id_producto es igual al id_producto que están ingresando a la tabla Incluye. Y finalmente este resultado es la tabla de donde la consulta principal va a obtener su resultado. Esta consulta principal nos devuelve cantidad_producto donde el id_producto en Producto es igual al id_producto que se está ingresando en la tabla Incluye. Así es como logramos obtener la suma de todos los cantidad_vendida de cada producto conforme se vaya agregando a una orden y se lo asignamos a la tabla Producto como su cantidad_vendida. En otras palabras, en este trigger calculamos la cantidad vendida de todos los productos en la base de datos, es decir, cuánto se han vendido.
   Si entra al IF y al IFELSE de cuando va a insertar o actualizar de la tabla Incluye se hace referencia a datos de los atributos a ingresar con un NEW. Si entra al IFELSE de cuando se va a borrar de la tabla Incluye, se hace referencia a los datos de los atributos a borrar con un OLD.

7. mesero_a_orden_trigger: Este trigger se implementa cuando se quiere insertar o actualizar en la tabla Orden. Lo que se realiza es dos cláusulas IF y ELSEIF. En la cláusula del IF va a entrar cuando es_mesero sea falso, es decir, que el numero_empleado que se intenta ingresar en la tabla Orden no sea mesero. Si es así, lanza un error que dice ."El empleado que se intenta asociar a la orden no es mesero". En la cláusula IFELSE hay una subconsulta que revisa si existe o no un empleado con el

numero_empleado que se está intentando ingresar a una orden. Si no existe, lanza un error que dice "El empleado que se intenta asociar a la orden no existe en la base de datos".

6.1.6. Funciones

Funciones implementadas:

1. folio_orden: Esta función no recibe como parámetros ningún valor, únicamente se declara una variable llamada identificador de tipo varchar con longitud 8, en la cual se almacena el folio con el formato requerido. Lo que se le asigna a esta variable es el valor de la concatenacion de la cadena 'ORD' con un casteo del número secuencial que fue implementado por un secuenciador, este secuenciador es utilizado por medio de un "SELECT nextval('nombre_secuenciador')" que nos va dando el siguiente valor, por lo que se incrementa cada vez que se ingrese una nueva orden, todo este casteo se convierte a varchar ya que es el tipo de dato definido para el folio de la orden. Para hacer uso de este se hace un llamado a la funcion dentro del insert en la orden con la siguiente sentencia: folio_orden().

2. mesero_ordenes_al_dia_bdd_function: En esta función se declara una variable llamada num_empleado de tipo int que es la que almacena el parámetro ingresado por el usuario. Se le indica que nos va a devolver una tabla conformada por numero_empleado de tipo int, numero_ordenes de tipo bigint y precio_total_ordenes de tipo real.
   La función tiene tres cláusulas que son IF, ELSEIF y ELSE. El IF verifica que si en empleado a buscar mediante su numero_empleado tiene FALSE en es_mesero, entonces arroja un error con el mensaje que dice "No es mesero". El ELSEIF verifica que exista el numero_empleado ingresado en la tabla Empleado y si es así, nos va a devolver la tabla la cual es el resultado de una consulta formada por subconsultas.
   La primera subconsulta nos va a devolver el total de registros de numero_empleado de la tabla orden en donde el numero_empleado en Orden sea igual al ingresado por el usuario, además se debe cumplir que sea de la fecha de este día, y para ello se utiliza la función EXTRACT para el día, el mes y el año. Se hacen comparaciones de la fecha en las ordenes y de la fecha de hoy. La segunda subconsulta nos devuelve una tabla con la suma de los precio_total de la tabla Orden donde el numero_empleado en Orden sea igual al ingresado por el usuario, además se debe cumplir que sea de la fecha de este día, y para ello se utiliza la función EXTRACT para el día, el mes y el año. Se hacen comparaciones de la fecha en las ordenes y de la fecha de hoy.
   Estos dos resultados de ambas subconsultas se van a mostrar como dos columnas en la consulta principal en la cual también se imprime el numero_empleado de la tabla Empleado donde el atributo es_mesero es TRUE y el numero_empleado sea igual al del parámetro de la función ingresado por el usuario. Además se utiliza un DISTINCT para evitar registros repetidos. Esta consulta principal nos devuelve cantidad_producto donde el id_producto en Producto es igual al id_producto que se está ingresando en la tabla Incluye. Finalmente en la cláusula ELSE se va a imprimir un mensaje de error que dice "No existe el empleado".
   Lo que lo hace diferente de la función mesero_ordenes_al_dia_function es que esta función descrita en este punto nos devuelve un error con RAISE EXCEPTION si no es mesero o no existe el empleado en la base de datos. En cambio, la función mesero_ordenes_al_dia_function es la que se implementa desde la interfaz y mediante el código en Python se pueda hacer una condición de que si la tabla que devuelve la función está vacía, significa que el empleado ingresado no es mesero o no existe en la base de datos.

3. mesero_ordenes_al_dia_function: En esta función se declara una variable llamada num_empleado de tipo int que es la que almacena el parámetro ingresado por el usuario. Se le indica que nos va a devolver una tabla conformada por numero_empleado de tipo int, numero_ordenes de tipo bigint y precio_total_ordenes de tipo real.
   La función tiene una cláusula IF. Esta cláusula IF verifica que exista el numero_empleado ingresado en la tabla Empleado y si es así, nos va a devolver la tabla la cual es el resultado de una consulta formada por subconsultas.
   La primera subconsulta nos va a devolver el total de registros de numero_empleado de la tabla orden en donde el numero_empleado en Orden sea igual al ingresado por el usuario, además se debe cumplir que sea de la fecha de este día, y para ello se utiliza la función EXTRACT para el día, el mes y el año. Se hacen comparaciones de la fecha en las ordenes y de la fecha de hoy.

La segunda subconsulta nos devuelve una tabla con la suma de los precio_total de la tabla Orden donde el numero_empleado en Orden sea igual al ingresado por el usuario, además se debe cumplir que sea de la fecha de este día, y para ello se utiliza la función EXTRACT para el día, el mes y el año. Se hacen comparaciones de la fecha en las ordenes y de la fecha de hoy.
Estos dos resultados de ambas subconsultas se van a mostrar como dos columnas en la consulta principal en la cual también se imprime el numero_empleado de la tabla Empleado donde el atributo es_mesero es TRUE y el numero_empleado sea igual al del parámetro de la función ingresado por el usuario. Además se utiliza un DISTINCT para evitar registros repetidos. Esta consulta principal nos devuelve cantidad_producto donde el id_producto en Producto es igual al id_producto que se está ingresando en la tabla Incluye. En otras palabras, esta función nos va a devolver la cantidad de ordenes de un mesero con el total del monto de dichas ordenes en el día de hoy.
Lo que lo hace diferente de la función mesero_ordenes_al_dia_bdd_function es que esta función descrita en este punto nos sirve para implementarla en la interfaz y que mediante el código en Python se pueda hacer una condición de que si la tabla que devuelve la función está vacía, significa que el empleado ingresado no es mesero o no existe en la base de datos. En cambio, la anterior función es la que devuelve un error con RAISE EXCEPTION si no es mesero o no existe el empleado en la base de datos.

4. cantidad_monto_ventas_fecha_bdd_function: En esta función se declara una variable llamada fecha_ingresada de tipo date que es la que almacena el parámetro ingresado por el usuario. Se le indica que nos va a devolver una tabla conformada por fecha de tipo date, cantidad_ordenes y cantidad_ventas de tipo bigint, y monto_total_ventas de tipo real.
   La función tiene dos cláusulas que son IF y ELSE. El IF verifica que exista fecha_ingresada en la tabla Orden y si es así, nos va a devolver la tabla la cual es el resultado de una consulta formada por subconsultas.
   La primera subconsulta nos va a devolver el total de registros de folio de la tabla Orden en donde la fecha en Orden sea igual a la ingresada por el usuario.
   La segunda subconsulta nos va a devolver la suma de cantidad_total de la tabla orden en donde la fecha en Orden sea igual a la ingresada por el usuario.
   La tercera subconsulta nos devuelve una tabla con la suma de los precio_total de la tabla Orden donde la fecha en Orden sea igual la ingresada por el usuario.
   Estos tres resultados de las tres subconsultas se van a mostrar como tres columnas en la consulta principal en la cual también se imprime la fecha de la tabla Orden donde la fecha en Orden sea igual la ingresada por el usuario. Se hace uso de DISTINCT para evitar registros repetidos. Finalmente en la cláusula ELSE se va a imprimir un mensaje de error que dice "No hay ventas en esa fecha".
   En otras palabras, dada una fecha ingresada por el usuario, nos va a devolver la cantidad de ordenes y ventas en esa fecha junto con el monto total de dichas ventas.
   Lo que lo hace diferente de la función cantidad_monto_ventas_fecha_function es que esta función descrita en este punto nos devuelve un error con RAISE EXCEPTION si no hay ventas almacenadas en esa fecha. En cambio, la función cantidad_monto_ventas_fecha_function es la que se implementa desde la interfaz y mediante el código en Python se pueda hacer una condición de que si la tabla que devuelve la función está vacía, significa que no hay ventas almacenadas en esa fecha.

5. cantidad_monto_ventas_fecha_function: En esta función se declara una variable llamada fecha_ingresada de tipo date que es la que almacena el parámetro ingresado por el usuario. Se le indica que nos va a devolver una tabla conformada por fecha de tipo date, cantidad_ordenes y cantidad_ventas de tipo bigint, y monto_total_ventas de tipo real.
   La función tiene una cláusula que es IF. Este IF verifica que exista fecha_ingresada en la tabla Orden y si es así, nos va a devolver la tabla la cual es el resultado de una consulta formada por subconsultas.
   La primera subconsulta nos va a devolver el total de registros de folio de la tabla Orden en donde la fecha en Orden sea igual a la ingresada por el usuario.
   La segunda subconsulta nos va a devolver la suma de cantidad_total de la tabla orden en donde la fecha en Orden sea igual a la ingresada por el usuario. La tercera subconsulta nos devuelve una tabla con la suma de los precio_total de la tabla Orden donde la fecha en Orden sea igual la ingresada por el usuario.
   Estos tres resultados de las tres subconsultas se van a mostrar como tres columnas en la consulta principal en la cual también se imprime la fecha de la tabla Orden donde la fecha en Orden sea igual la

ingresada por el usuario. Se hace uso de DISTINCT para evitar registros repetidos. En otras palabras, dada una fecha ingresada por el usuario, nos va a devolver la cantidad de ordenes y ventas en esa fecha junto con el monto total de dichas ordenes.
Lo que lo hace diferente de la función cantidad_monto_ventas_fecha_bdd_function es que esta función descrita en este punto nos sirve para implementarla en la interfaz y que mediante el código en Python se pueda hacer una condición de que si la tabla que devuelve la función está vacía, significa que no hay ventas en esa fecha. En cambio, la anterior función es la que devuelve un error con RAISE EXCEPTION si no hay ventas en la fecha ingresada por el usuario.

6. cantidad_monto_ventas_fechas_bdd_function: En esta función se declaran dos variables llamadas fecha_de_inicio y fecha_de_fin de tipo date que son las que almacenan a los parámetros ingresados por el usuario. Se le indica que nos va a devolver una tabla conformada por fecha_inicio y fecha_fin de tipo date, cantidad_ordenes y cantidad_ventas de tipo bigint, y monto_total_ventas de tipo real.
   La función tiene dos cláusulas que son IF y ELSE. El IF verifica que exista una fecha en la tabla Orden que esté dentro del rango de fecha_de_inicio y fecha_de_fin haciendo uso del operador lógico BETWEEN y si es así, nos va a devolver la tabla la cual es el resultado de una consulta formada por subconsultas.
   La primera subconsulta nos va a devolver el total de registros de folio de la tabla Orden en donde la fecha en Orden se encuentre entre el rango de fecha_de_inicio y fecha_de_fin.
   La segunda subconsulta nos va a devolver la suma de cantidad_total de la tabla Orden en donde la fecha en Orden se encuentre entre el rango de fecha_de_inicio y fecha_de_fin.
   La tercera subconsulta nos devuelve una tabla con la suma de los precio_total de la tabla Orden donde la fecha en Orden se encuentre entre el rango de fecha_de_inicio y fecha_de_fin.
   Estos tres resultados de estas tres subconsultas se van a mostrar como tres columnas en la consulta principal en la cual también se imprime fecha_de_inicio y fecha_de_fin. Se hace uso de DISTINCT para evitar registros repetidos.
   Finalmente en la cláusula ELSE se va a imprimir un mensaje de error que dice "No hay ventas entre esas fechas". En otras palabras, dadas dos fechas ingresadas por el usuario, nos va a devolver la cantidad de ordenes y ventas entre esas fechas junto con el monto total de dichas ventas.
   Lo que lo hace diferente de la función cantidad_monto_ventas_fechas_function es que esta función descrita devuelve un error con RAISE EXCEPTION si no hay ventas almacenadas entre esas fechas. En cambio, la función cantidad_monto_ventas_fechas_function es la que se implementa desde la interfaz y mediante el código en Python se pueda hacer una condición de que si la tabla que devuelve la función está vacía, significa que no hay ventas almacenadas entre esas fechas.

7. cantidad_monto_ventas_fechas_function: En esta función se declaran dos variables llamadas fecha_de_inicio y fecha_de_fin de tipo date que son las que almacenan a los parámetros ingresados por el usuario. Se le indica que nos va a devolver una tabla conformada por fecha_inicio y fecha_fin de tipo date, cantidad_ordenes y cantidad_ventas de tipo bigint, y monto_total_ventas de tipo real.
   La función tiene una cláusula que es IF. Este IF verifica que exista una fecha en la tabla Orden que esté dentro del rango de fecha_de_inicio y fecha_de_fin haciendo uso del operador lógico BETWEEN y si es así, nos va a devolver la tabla la cual es el resultado de una consulta formada por subconsultas.
   La primera subconsulta nos va a devolver el total de registros de folio de la tabla Orden en donde la fecha en Orden se encuentre entre el rango de fecha_de_inicio y fecha_de_fin.
   La segunda subconsulta nos va a devolver la suma de cantidad_total de la tabla Orden en donde la fecha en Orden se encuentre entre el rango de fecha_de_inicio y fecha_de_fin.
   La tercera subconsulta nos devuelve una tabla con la suma de los precio_total de la tabla Orden donde la fecha en Orden se encuentre entre el rango de fecha_de_inicio y fecha_de_fin. Estos tres resultados de estas tres subconsultas se van a mostrar como tres columnas en la consulta principal en la cual también se imprime fecha_de_inicio y fecha_de_fin. Se hace uso de DISTINCT para evitar registros repetidos.
   En otras palabras, dadas dos fechas ingresadas por el usuario, nos va a devolver la cantidad de ordenes y ventas entre esas fechas junto con el monto total de dichas ventas.
   Lo que lo hace diferente de la función cantidad_monto_ventas_fechas_bdd_function es que esta función descrita en este punto nos sirve para implementarla en la interfaz y que mediante el código en Python se pueda hacer una condición de que si la tabla que devuelve la función está vacía, significa que no hay ventas entre esas fechas. En cambio, la anterior función es la que devuelve un error con RAISE EXCEPTION si no hay ventas entre las fechas ingresadas por el usuario.

8. factura: Esta función recibe el folio de la orden del cliente y genera una visualización de su factura. Los elementos que componen la factura son:

   - Razón social
   - Folio de la orden
   - Fecha de la orden
   - Nombre del cliente
   - Dirección del cliente
   - RFC del cliente
   - Email del cliente
   - Cantidad de producto
   - Descripción del producto
   - Precio del producto
   - Importe por producto
   - Cantidad total del productos
   - Importe total por la orden

   Debido a que cada uno de estos valores se encuentra en tablas diferentes, se realizaron varios join que enlazarán las tablas que contienen dichos valores. En la tabla cliente, se encuentran los datos del cliente, en la tabla orden se ubican los datos de la orden, en la tabla incluye se especifican los productos llevados por la orden y en la tabla producto se encuentran los datos referentes al producto. Realizando un join que una cada una de estas tablas, es posible obtener una tabla resultante que contiene cada uno de los datos que muestra la factura. Una vez obtenida una tabla que tuviera por columnas cada uno de los datos de la factura, se declaró un cursor que recorriera la tabla. Este cursor recorre para cada una de las filas que corresponde al nombre y precio del producto, así como la cantidad del producto y el precio total por producto. Se propuso utilizar un cursor porque un cliente puede llevar varios productos, en diferentes cantidades y precios, por lo que cursor recorre cada una de las filas para conocer cada conocer la descripción de cada uno de los productos de la orden. Cabe decir que, los valores de las filas fueron registrados con variables de tipo RECORD. La visualización de la factura se muestra a continuación.

```
NOTICE:  - - -  RESTAURANTES COPACABANA - - -
NOTICE:
NOTICE:  Folio de la orden: ORD-002
NOTICE:  Fecha: 2022-08-13
NOTICE:  FACTURAR A Fernando Garcia Rios
NOTICE:  Dirección: Eje Central 59 Haciendas De Coyoacan 4970 Durango
NOTICE:  RFC: GARF941010
NOTICE:  Email: Fernando.Garc@correo.com
NOTICE:
NOTICE:  Cantidad|Descripcion    |Precio    |Importe
NOTICE:    2     | Huevos revueltos con tocino         | 90.5     | 181
NOTICE:    1     | Cerveza         | 60.5     | 60.5
NOTICE:
NOTICE:  Cantidad total de productos: 3
NOTICE:  Importe total: 241.5

Successfully run. Total query runtime: 34 msec.
1 rows affected.
```

Figura 3: Factura

6.1.7. Vistas

Vistas implementadas:

1. platillo_mas_vendido_view: La consulta de esta vista está compuesta también por una subconsulta. La subconsulta nos devuelve el valor máximo de cantidad_vendida de la tabla Producto en donde el tipo_producto sea igual a "Platillo". Este resultado es de donde la consulta principal va a obtener sus datos. La consulta principal imprime toda la información en la tabla Producto donde cantidad_vendida sea igual al resultado de la subconsulta.

2. productos_no_disponibles_view: La consulta de esta vista está compuesta por una consulta que te regresa el nombre de los productos en la tabla Productos que tienen en su atributo disponibilidad un FALSE.

6.2. Código final

6.2.1. Tablas

Código de cada una de las tablas:

Tabla empleado

```
CREATE TABLE empleado
(
    numero_empleado integer NOT NULL,
    foto varchar(500) NOT NULL,
    rfc character varying(13) NOT NULL,
    nombre character varying(60) NOT NULL,
    ap_paterno character varying(60) NOT NULL,
    ap_materno character varying(60) NULL,
    edad smallint NULL,
    sueldo float NOT NULL,
    calle character varying(80) NOT NULL,
    numero smallint NOT NULL,
    cp integer NOT NULL,
    estado character varying(60) NOT NULL,
    colonia character varying(100) NOT NULL,
    fecha_nacimiento date NOT NULL,
    es_mesero boolean NOT NULL,
    es_administrativo boolean NOT NULL,
    es_cocinero boolean NOT NULL,
    horario character varying(60) NULL default 'No aplica',
    rol character varying(60) NULL default 'No aplica',
    especialidad character varying(60) NULL default 'No aplica',
    CONSTRAINT pk_empleado PRIMARY KEY (numero_empleado),
    CONSTRAINT ak_rfc UNIQUE (rfc)
);
```

Figura 4

Tabla telefono

```
CREATE TABLE telefono
(
    telefono bigint NOT NULL,
    numero_empleado integer NOT NULL,
    CONSTRAINT pk_telefono PRIMARY KEY (telefono),
    CONSTRAINT fk_telefono_empleado FOREIGN KEY (numero_empleado) REFERENCES
    empleado (numero_empleado) ON UPDATE CASCADE ON DELETE CASCADE
);
```

Figura 5

Tabla dependiente

```
CREATE TABLE dependiente
(
    curp character varying(18) NOT NULL,
    nombre character varying(60) NOT NULL,
    ap_paterno character varying(60) NOT NULL,
    ap_materno character varying(60) NULL,
    parentesco character varying(30) NOT NULL,
    numero_empleado integer NOT NULL,
    CONSTRAINT pk_dependiente PRIMARY KEY (curp),
    CONSTRAINT fk_dependiente_empleado FOREIGN KEY (numero_empleado) REFERENCES
    empleado (numero_empleado) ON UPDATE CASCADE ON DELETE CASCADE
);
```

Figura 6

Tabla cliente

```
CREATE TABLE cliente
(
    rfc character varying(13) NOT NULL,
    email character varying(200) NOT NULL,
    nombre character varying(60) NOT NULL,
    ap_paterno character varying(60) NOT NULL,
    ap_materno character varying(60) NULL,
    fecha_nacimiento date NOT NULL,
    razon_social character varying(150) NOT NULL,
    cp integer NOT NULL,
    estado character varying(60) NOT NULL,
    numero smallint NOT NULL,
    calle character varying(80) NOT NULL,
    colonia character varying(100) NOT NULL,
    CONSTRAINT pk_cliente PRIMARY KEY (rfc)
);
```

Figura 7

Tabla orden

```
CREATE TABLE orden
(
    folio character varying(20) NOT NULL,
    fecha date NOT NULL default now(),
    precio_total real NULL,
    cantidad_total smallint NULL,
    numero_empleado integer NULL,
    rfc character varying(13) NULL,
    CONSTRAINT pk_orden PRIMARY KEY (folio),
    CONSTRAINT fk_orden_cliente FOREIGN KEY (rfc) REFERENCES cliente (rfc)
    ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT fk_orden_empleado FOREIGN KEY (numero_empleado) REFERENCES
    empleado (numero_empleado) ON UPDATE CASCADE ON DELETE SET NULL
);
```

Figura 8

Tabla categoria

```
CREATE TABLE categoria
(
    id_categoria integer NOT NULL,
    nombre character varying(50)  NOT NULL,
    descripcion character varying(500) NOT NULL,
    CONSTRAINT pk_categoria PRIMARY KEY (id_categoria)
);
```

Figura 9

Tabla producto

```
CREATE TABLE producto
(
    id_producto integer NOT NULL,
    nombre character varying(150) NOT NULL,
    receta character varying(2000) NOT NULL,
    precio real NOT NULL,
    descripcion character varying(400) NOT NULL,
    disponibilidad boolean NOT NULL,
    tipo_producto character varying(8) NOT NULL,
    con_alcohol boolean NULL,
    platillo_del_dia boolean NULL,
    id_categoria integer NULL,
    cantidad_vendida integer NULL,
    CONSTRAINT pk_producto PRIMARY KEY (id_producto),
    CONSTRAINT fk_producto_categoria FOREIGN KEY (id_categoria) REFERENCES
    categoria (id_categoria) ON UPDATE CASCADE ON DELETE SET NULL,
    CONSTRAINT ck_precio CHECK (precio > 0)
);
```

Figura 10

Tabla incluye

```
CREATE TABLE incluye
(
    folio character varying(20) NOT NULL,
    id_producto integer NOT NULL,
    cantidad_producto integer NOT NULL,
    precio_total_por_producto float NULL,
    CONSTRAINT pk_incluye PRIMARY KEY (folio, id_producto),
    CONSTRAINT fk_incluye_orden FOREIGN KEY (folio) REFERENCES orden (folio)
    ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_incluye_producto FOREIGN KEY (id_producto) REFERENCES
    producto (id_producto) ON UPDATE CASCADE ON DELETE RESTRICT,
    CONSTRAINT ck_cantidad_producto CHECK (cantidad_producto > 0)
);
```

Figura 11

6.2.2. Secuenciadores

Código del secuenciador:

Secuenciador folio

```
CREATE SEQUENCE folio
START WITH 00001
INCREMENT BY 1
MAXVALUE 99999
MINVALUE 00001
NO CYCLE;
```

Figura 12

6.2.3. Índices

Código del índice:

```
CREATE INDEX ix_nombre_producto
ON producto USING btree (nombre);
```

Figura 13

6.2.4. Disparadores

Código de los disparadores:

edad_trigger()

```
CREATE FUNCTION edad_trigger() RETURNS TRIGGER AS $edad_trigger$
BEGIN
NEW.edad=date_part('year',age(now(),NEW.fecha_nacimiento));
RETURN NEW;
END;
$edad_trigger$ LANGUAGE plpgsql;
CREATE TRIGGER edad_trigger BEFORE INSERT OR UPDATE ON empleado FOR EACH ROW EXECUTE PROCEDURE edad_trigger();
```

Figura 14: Trigger que calcula la edad

cantidad_total_trigger()

```
CREATE FUNCTION cantidad_total_trigger() RETURNS TRIGGER AS $cantidad_total$
BEGIN
IF (TG_OP ='INSERT') THEN
    UPDATE orden SET cantidad_total=(SELECT cantidad_producto from (SELECT cantidad_producto from (SELECT folio, sum(cantidad_producto) as
cantidad_producto from incluye group by folio) as suma WHERE folio=NEW.folio) as resultado) WHERE orden.folio=NEW.folio;
ELSEIF (TG_OP ='UPDATE') THEN
    UPDATE orden SET cantidad_total=(SELECT cantidad_producto from (SELECT cantidad_producto from (SELECT folio, sum(cantidad_producto) as
cantidad_producto from incluye group by folio) as suma WHERE folio=NEW.folio) as resultado) WHERE orden.folio=NEW.folio;
ELSEIF (TG_OP ='DELETE') THEN
    UPDATE orden SET cantidad_total=(SELECT cantidad_producto from (SELECT cantidad_producto from (SELECT folio, sum(cantidad_producto) as
cantidad_producto from incluye group by folio) as suma WHERE folio=OLD.folio) as resultado) WHERE orden.folio=OLD.folio;
END IF;
RETURN NULL;
END;
$cantidad_total$ LANGUAGE plpgsql;
CREATE TRIGGER cantidad_total_ins_trigger AFTER INSERT ON incluye FOR EACH ROW EXECUTE PROCEDURE cantidad_total_trigger();
CREATE TRIGGER cantidad_total_upd_trigger AFTER UPDATE ON incluye FOR EACH ROW EXECUTE PROCEDURE cantidad_total_trigger();
CREATE TRIGGER cantidad_total_del_trigger AFTER DELETE ON incluye FOR EACH ROW EXECUTE PROCEDURE cantidad_total_trigger();
```

Figura 15: Trigger que calcula la cantidad total de productos de la orden

precio_total_por_producto_trigger()

```
CREATE FUNCTION precio_total_por_producto_trigger() RETURNS TRIGGER AS $precio_total_por_producto_trigger$
BEGIN
NEW.precio_total_por_producto=(NEW.cantidad_producto)*(SELECT precio from producto WHERE id_producto=NEW.id_producto);
RETURN NEW;
END;
$precio_total_por_producto_trigger$ LANGUAGE plpgsql;
CREATE TRIGGER precio_total_por_producto_trigger BEFORE INSERT OR UPDATE ON incluye FOR EACH ROW EXECUTE PROCEDURE
precio_total_por_producto_trigger();
```

Figura 16: Trigger que calcula el precio total por productos de la orden

precio_total_trigger()

```
CREATE FUNCTION precio_total_trigger() RETURNS TRIGGER AS $precio_total$
BEGIN
IF (TG_OP ='INSERT') THEN
    UPDATE orden SET precio_total=(SELECT precio_total_por_producto from (SELECT precio_total_por_producto from (SELECT folio,
sum(precio_total_por_producto) as precio_total_por_producto from incluye group by folio) as suma WHERE folio=NEW.folio) as resultado) WHERE
orden.folio=NEW.folio;
ELSEIF (TG_OP ='UPDATE') THEN
    UPDATE orden SET precio_total=(SELECT precio_total_por_producto from (SELECT precio_total_por_producto from (SELECT folio,
sum(precio_total_por_producto) as precio_total_por_producto from incluye group by folio) as suma WHERE folio=NEW.folio) as resultado) WHERE
orden.folio=NEW.folio;
ELSEIF (TG_OP ='DELETE') THEN
    UPDATE orden SET precio_total=(SELECT precio_total_por_producto from (SELECT precio_total_por_producto from (SELECT folio,
sum(precio_total_por_producto) as precio_total_por_producto from incluye group by folio) as suma WHERE folio=OLD.folio) as resultado) WHERE
orden.folio=OLD.folio;

END IF;
RETURN NULL;
END;
$precio_total$ LANGUAGE plpgsql;
CREATE TRIGGER precio_total_ins_trigger AFTER INSERT ON incluye FOR EACH ROW EXECUTE PROCEDURE precio_total_trigger();
CREATE TRIGGER precio_total_upd_trigger AFTER UPDATE ON incluye FOR EACH ROW EXECUTE PROCEDURE precio_total_trigger();
CREATE TRIGGER precio_total_del_trigger AFTER DELETE ON incluye FOR EACH ROW EXECUTE PROCEDURE precio_total_trigger();
```

Figura 17: Trigger que calcula el precio total de la orden

cantidad_vendida_trigger()

```
CREATE FUNCTION cantidad_vendida_trigger() RETURNS TRIGGER AS $cantidad_vendida$
BEGIN
IF (TG_OP ='INSERT') THEN
    UPDATE producto SET cantidad_vendida=(SELECT cantidad_producto from (SELECT cantidad_producto from (SELECT id_producto, sum(cantidad_producto)
as cantidad_producto from incluye group by id_producto) as suma WHERE id_producto=NEW.id_producto) as resultado) WHERE
producto.id_producto=NEW.id_producto;
ELSEIF (TG_OP ='UPDATE') THEN
    UPDATE producto SET cantidad_vendida=(SELECT cantidad_producto from (SELECT cantidad_producto from (SELECT id_producto, sum(cantidad_producto)
as cantidad_producto from incluye group by id_producto) as suma WHERE id_producto=NEW.id_producto) as resultado) WHERE
producto.id_producto=NEW.id_producto;
ELSEIF (TG_OP ='DELETE') THEN
    UPDATE producto SET cantidad_vendida=(SELECT cantidad_producto from (SELECT cantidad_producto from (SELECT id_producto, sum(cantidad_producto)
as cantidad_producto from incluye group by id_producto) as suma WHERE id_producto=OLD.id_producto) as resultado) WHERE
producto.id_producto=OLD.id_producto;
END IF;
RETURN NULL;
END;
$cantidad_vendida$ LANGUAGE plpgsql;
CREATE TRIGGER cantidad_vendida_ins_trigger AFTER INSERT ON incluye FOR EACH ROW EXECUTE PROCEDURE cantidad_vendida_trigger();
CREATE TRIGGER cantidad_vendida_upd_trigger AFTER UPDATE ON incluye FOR EACH ROW EXECUTE PROCEDURE cantidad_vendida_trigger();
CREATE TRIGGER cantidad_vendida_del_trigger AFTER DELETE ON incluye FOR EACH ROW EXECUTE PROCEDURE cantidad_vendida_trigger();
```

Figura 18: Trigger que calcula la cantidad vendida de un producto

disponibilidad_producto_trigger()

```
CREATE FUNCTION disponibilidad_producto_trigger() RETURNS TRIGGER AS $disponibilidad_producto_trigger$
BEGIN
IF ((SELECT disponibilidad from producto WHERE id_producto=NEW.id_producto)=false) THEN
    RAISE EXCEPTION 'Producto no disponible por el momento';
END IF;
RETURN NEW;
END;
$disponibilidad_producto_trigger$ LANGUAGE plpgsql;
CREATE TRIGGER disponibilidad_producto_trigger BEFORE INSERT OR UPDATE ON incluye FOR EACH ROW EXECUTE PROCEDURE
disponibilidad_producto_trigger();
```

Figura 19: Trigger que nos avisa si un producto no esta disponible

mesero_a_orden_trigger()

```
CREATE FUNCTION mesero_a_orden_trigger() RETURNS TRIGGER AS $mesero_a_orden_trigger$
BEGIN
IF ((SELECT es_mesero from empleado WHERE numero_empleado=NEW.numero_empleado)=false) THEN
    RAISE EXCEPTION 'El empleado que se intenta asociar a la orden no es mesero';
ELSEIF (NOT EXISTS(SELECT numero_empleado from empleado WHERE numero_empleado=NEW.numero_empleado)) THEN
    RAISE EXCEPTION 'El empleado que se intenta asociar a la orden no existe en la base de datos';
END IF;
RETURN NEW;
END;
$mesero_a_orden_trigger$ LANGUAGE plpgsql;
CREATE TRIGGER mesero_a_orden_trigger BEFORE INSERT OR UPDATE ON orden FOR EACH ROW EXECUTE PROCEDURE mesero_a_orden_trigger();
```

Figura 20: Trigger que asocia a un solo mesero a una orden

6.2.5. Funciones

Código de las funciones:

folio_orden()

```
CREATE OR REPLACE FUNCTION folio_orden() RETURNS VARCHAR AS $folio_orden$
DECLARE identificador varchar(8);
BEGIN
identificador:=CONCAT('ORD-',CAST((SELECT nextval('folio')) AS VARCHAR));
RETURN identificador;
END;
$folio_orden$  LANGUAGE plpgsql;
```

Figura 21: Función que genera el folio

mesero_ordenes_al_dia_bdd_function(num_empleado int)

```
CREATE OR REPLACE FUNCTION mesero_ordenes_al_dia_bdd_function
(
    num_empleado int
)
returns table(numero_empleado int,numero_ordenes bigint, precio_total_ordenes real)
LANGUAGE plpgsql AS
$func$
BEGIN
    IF((SELECT es_Mesero from empleado WHERE empleado.numero_empleado=num_empleado)=FALSE) THEN
        RAISE EXCEPTION 'No es mesero';
    ELSEIF (EXISTS(SELECT empleado.numero_empleado from empleado WHERE empleado.numero_empleado=num_empleado)) THEN
        return query
        SELECT DISTINCT empleado.numero_empleado,(select count(orden.numero_empleado) as numero_ordenes from orden WHERE
orden.numero_empleado=num_empleado and extract(year from fecha)=extract(year from now()) and extract(month from fecha)=extract(month from now())
and extract(day from fecha)=extract(day from now())),(select sum(orden.precio_total) as precio_total_ordenes from orden WHERE
orden.numero_empleado=num_empleado and extract(year from fecha)=extract(year from now()) and extract(month from fecha)=extract(month from now())
and extract(day from fecha)=extract(day from now())) from empleado WHERE es_mesero=TRUE AND empleado.numero_empleado=num_empleado;
    ELSE
        RAISE EXCEPTION 'No existe el empleado';
   END IF;
END
$func$;
```

Figura 22: Función que calcula cuantas ordenes ha realizado un mesero ese día desde la base de datos

mesero_ordenes_al_dia_function (num_empleado int)

```
CREATE OR REPLACE FUNCTION mesero_ordenes_al_dia_function
(
    num_empleado int
)
returns table(numero_empleado int,numero_ordenes bigint, precio_total_ordenes real)
LANGUAGE plpgsql AS
$func$
BEGIN
    IF (EXISTS(SELECT empleado.numero_empleado from empleado WHERE empleado.numero_empleado=num_empleado) AND (SELECT es_Mesero from empleado WHERE
empleado.numero_empleado=num_empleado)=TRUE) THEN
        return query
        SELECT DISTINCT empleado.numero_empleado,(select count(orden.numero_empleado) as numero_ordenes from orden WHERE
orden.numero_empleado=num_empleado and extract(year from fecha)=extract(year from now()) and extract(month from fecha)=extract(month from now())
and extract(day from fecha)=extract(day from now())),(select sum(orden.precio_total) as precio_total_ordenes from orden WHERE
orden.numero_empleado=num_empleado and extract(year from fecha)=extract(year from now()) and extract(month from fecha)=extract(month from now())
and extract(day from fecha)=extract(day from now())) from empleado WHERE es_mesero=TRUE AND empleado.numero_empleado=num_empleado;
    END IF;

END
$func$;
```

Figura 23: Función que calcula cuantas ordenes ha realizado un mesero ese día desde la interfaz

cantidad_monto_ventas_fecha_bdd_function(fecha_ingresada date)

```
CREATE OR REPLACE FUNCTION cantidad_monto_ventas_fecha_bdd_function
(
    fecha_ingresada date
)
returns table(fecha date, cantidad_ordenes bigint, cantidad_ventas bigint,monto_total_ventas real)
LANGUAGE plpgsql AS
$func$
BEGIN
    IF (EXISTS(SELECT orden.fecha from orden WHERE orden.fecha=fecha_ingresada)) THEN
        return query
        SELECT DISTINCT orden.fecha,(select count(orden.folio) as cantidad_ordenes from orden WHERE orden.fecha=fecha_ingresada),(select
sum(orden.cantidad_total) as cantidad_ventas from orden WHERE orden.fecha=fecha_ingresada),(select sum(orden.precio_total) as monto_total_ventas
from orden WHERE orden.fecha=fecha_ingresada) from orden WHERE orden.fecha=fecha_ingresada;
    ELSE
        RAISE EXCEPTION 'No hay ventas en esa fecha';
   END IF;

END
$func$;
```

Figura 24: Función calcula el monto de las ventas dada una sola fecha desde la base de datos

cantidad_monto_ventas_fecha_function(fecha_ingresada date)

```
CREATE OR REPLACE FUNCTION cantidad_monto_ventas_fecha_function
(
    fecha_ingresada date
)
returns table(fecha date, cantidad_ordenes bigint, cantidad_ventas bigint,monto_total_ventas real)
LANGUAGE plpgsql AS
$func$
BEGIN
    IF (EXISTS(SELECT orden.fecha from orden WHERE orden.fecha=fecha_ingresada)) THEN
        return query
        SELECT DISTINCT orden.fecha,(select count(orden.folio) as cantidad_ordenes from orden WHERE orden.fecha=fecha_ingresada),(select
sum(orden.cantidad_total) as cantidad_ventas from orden WHERE orden.fecha=fecha_ingresada),(select sum(orden.precio_total) as monto_total_ventas
from orden WHERE orden.fecha=fecha_ingresada) from orden WHERE orden.fecha=fecha_ingresada;

   END IF;

END
$func$;
```

Figura 25: Función calcula el monto de las ventas dada una sola fecha desde la interfaz

cantidad_monto_ventas_fechas_bdd_function(fecha_de_inicio date, fecha_de_fin date)

```
CREATE OR REPLACE FUNCTION cantidad_monto_ventas_fechas_bdd_function
(
    fecha_de_inicio date,
    fecha_de_fin date
)
returns table(fecha_inicio date, fecha_fin date,cantidad_ordenes bigint,cantidad_ventas bigint,monto_total_ventas real)
LANGUAGE plpgsql AS
$func$
BEGIN
    IF (EXISTS(SELECT orden.fecha from orden WHERE orden.fecha BETWEEN fecha_de_inicio AND fecha_de_fin)) THEN
        return query
        SELECT DISTINCT fecha_de_inicio,fecha_de_fin,(select count(orden.folio) as cantidad_ordenes from orden WHERE orden.fecha BETWEEN
fecha_de_inicio AND fecha_de_fin),(select sum(orden.cantidad_total) as cantidad_ventas from orden WHERE orden.fecha BETWEEN fecha_de_inicio AND
fecha_de_fin),(select sum(orden.precio_total) as monto_total_ventas from orden WHERE orden.fecha BETWEEN fecha_de_inicio AND fecha_de_fin) from
orden;
    ELSE
        RAISE EXCEPTION 'No hay ventas entre esas fechas';
   END IF;

END
$func$;
```

Figura 26: Función que calcula el monto de las ventas dadas 2 fechas desde la base de datos

cantidad_monto_ventas_fechas_function(fecha_de_inicio date, fecha_de_fin date)

```
CREATE OR REPLACE FUNCTION cantidad_monto_ventas_fechas_function
(
    fecha_de_inicio date,
    fecha_de_fin date
)
returns table(fecha_inicio date, fecha_fin date, cantidad_ordenes bigint,cantidad_ventas bigint,monto_total_ventas real)
LANGUAGE plpgsql AS
$func$
BEGIN
    IF (EXISTS(SELECT orden.fecha from orden WHERE orden.fecha BETWEEN fecha_de_inicio AND fecha_de_fin)) THEN
        return query
        SELECT DISTINCT fecha_de_inicio,fecha_de_fin,(select count(orden.folio) as cantidad_ordenes from orden WHERE orden.fecha BETWEEN
fecha_de_inicio AND fecha_de_fin),(select sum(orden.cantidad_total) as cantidad_ventas from orden WHERE orden.fecha BETWEEN fecha_de_inicio AND
fecha_de_fin),(select sum(orden.precio_total) as monto_total_ventas from orden WHERE orden.fecha BETWEEN fecha_de_inicio AND fecha_de_fin) from
orden;

   END IF;

END
$func$;
```

Figura 27: Función que calcula el monto de las ventas dadas 2 fechas desde la interfaz

factura(folio_Orden varchar)

```
CREATE OR REPLACE FUNCTION factura(folio_Orden varchar) RETURNS void AS
$$
DECLARE
        registro RECORD;
        registro2 RECORD;
        cur_producto CURSOR FOR
        SELECT
                orden.folio AS Folio,
        incluye.id_producto AS idProducto,
                producto.nombre AS Producto,
                producto.precio AS Precio,
        incluye.cantidad_producto AS Cantidad,
        incluye.precio_total_por_producto AS PrecioPorProducto
   FROM cliente
     JOIN orden ON cliente.rfc::text = orden.rfc::text
     JOIN incluye ON orden.folio::text = incluye.folio::text
     JOIN producto ON incluye.id_producto = producto.id_producto
WHERE folio_Orden = orden.folio;

BEGIN
        SELECT UPPER(cliente.razon_social) AS RazonSocial,
    cliente.rfc AS RFC,
    concat(cliente.calle, ' ', cliente.numero, ' ', cliente.colonia, ' ', cliente.cp, ' ', cliente.estado, ' ') AS Direccion,
    concat(cliente.nombre, ' ', cliente.ap_paterno, ' ', cliente.ap_materno, ' ') AS Nombre,
    cliente.email AS Email,
    cliente.fecha_nacimiento AS FechaNacimiento,
    orden.folio AS Folio,
    orden.fecha AS Fecha,
    orden.cantidad_total AS CantidadTotal,
    orden.precio_total AS PrecioTotal
   INTO registro
   FROM cliente
     JOIN orden ON cliente.rfc::text = orden.rfc::text
     JOIN incluye ON orden.folio::text = incluye.folio::text
     JOIN producto ON incluye.id_producto = producto.id_producto
WHERE folio_Orden = orden.folio
GROUP BY cliente.rfc, orden.folio;
```

Figura 28: Función que genera la factura

```
-- Encabezado de la factura
        RAISE NOTICE '- - -  % - - -', registro.RazonSocial;
        RAISE NOTICE '';
        RAISE NOTICE 'Folio de la orden: %', registro.Folio;
        RAISE NOTICE 'Fecha: %', registro.Fecha;
        RAISE NOTICE 'FACTURAR A %', registro.Nombre;
        RAISE NOTICE 'Dirección: %', registro.Direccion;
        RAISE NOTICE 'RFC: %', registro.RFC;
        RAISE NOTICE 'Email: %', registro.Email;


-- Cuerpo de la factura
        OPEN cur_producto;
        FETCH cur_producto INTO registro2;

        RAISE NOTICE '';
        RAISE NOTICE 'Cantidad|Descripcion    |Precio    |Importe';

        WHILE(registro2.Folio = folio_Orden) LOOP
                RAISE NOTICE '  %     | %         | %     | %', registro2.Cantidad, registro2.Producto, registro2.Precio,
registro2.PrecioPorProducto;
                FETCH cur_producto INTO registro2;
        END LOOP;

        CLOSE cur_producto;

-- Fin de la factura
        RAISE NOTICE '';
        RAISE NOTICE 'Cantidad total de productos: %', registro.CantidadTotal;
        RAISE NOTICE 'Importe total: %', registro.PrecioTotal;
END
$$ LANGUAGE 'plpgsql';
```

Figura 29: Cuerpo de la factura

6.2.6. Vistas

Código de las vistas:
platillo_mas_vendido_view

```
CREATE VIEW platillo_mas_vendido_view AS SELECT * from producto WHERE cantidad_vendida=(SELECT MAX(cantidad_vendida)
from producto WHERE tipo_producto='Platillo') AND tipo_producto='Platillo';
```

Figura 30: Vista del platillo más vendido

productos_no_disponibles_view

```
CREATE VIEW productos_no_disponibles_view AS
SELECT nombre AS "Productos no disponibles"
FROM producto
WHERE disponibilidad=FALSE;
```

Figura 31: Vista de los productos no disponibles

7. Presentación

Decidimos hacer uso del lenguaje Python debido a que los conocimientos que tenemos respecto a programación web o móvil son muy pocos.

Se decidió trabajar la parte gráfica separando sus diferentes objetivos, ya que para la implementación de nuestro proyecto se requería una interfaz destinada a la parte de la administración del restaurante y otra para el ingreso de ordenes, pensada para el uso de los meseros. Para la interfaz se hizo uso de la biblioteca gráfica tkinter que viene instalada por defecto con la instalación de Python 3, la cual nos fue de gran apoyo para lograr implementar cada una de las funcionalidades necesarias, ya que es una librería con bastantes elementos para realizar diferentes tareas.

La parte de la conexión con PostgreSQL la realizamos a través del adaptador psycopg2, este adaptador es bastante simple y su facilidad permite inferir como funciona. Psycopg2 permite ejecutar sentencias SQL, respetando la sintaxis que posee PostgreSQL, es decir, es como si escribiéramos directamente en su terminal.

Una de los requisitos que más nos costó trabajo realizar, fue la visualización de la fotografía de los empleados dentro de la interfaz. Este problema fue resuelto utilizando el módulo urllib.request.

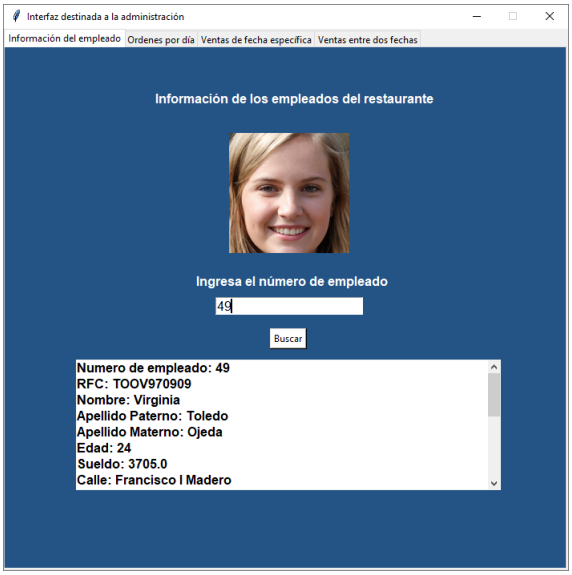
Urllib.request es un módulo de Python que se utiliza para acceder y utilizar recursos de internet identificados por URLs. Ofrece una interfaz muy simple, a través de la función urlopen. Esta función es capaz de acceder a URLs usando una variedad de protocolos diferentes. Una vez abierto el URL respectivo de la foto de cada empleado, se recupera dentro de Python con el método read() y posteriormente se ejecuta el método close(). Posteriormente se hace uso de la biblioteca PIL y de los módulos Image e ImageTK, para colocar la fotografía dentro de un label en la interfaz.

A continuación se muestran algunas capturas de la interfaz destinada a la parte de la administración del restaurante:

La primer pestaña solicita el número de empleado para desplegar su información y su fotografía.

De acuerdo a la información que se ingrese en el recuadro se pueden presentar las siguientes situaciones:

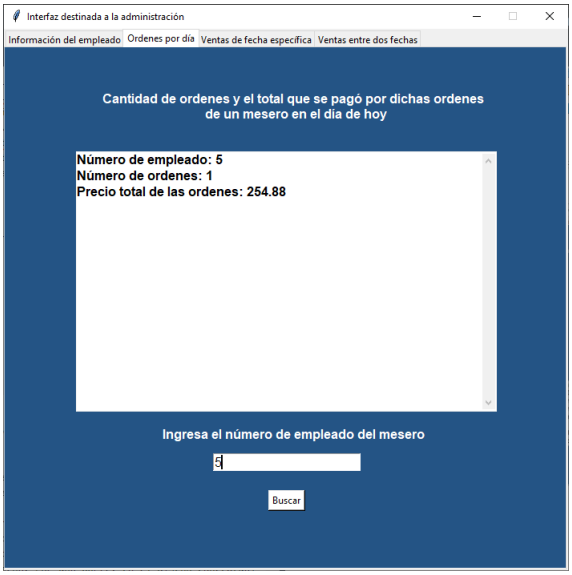
- Si el usuario ingresa un número y el empleado existe, se muestra su información y su fotografía.
- Si el usuario ingresa un número y no existe el empleado se muestra: "El empleado no existe".
- Si el usuario oprime el botón buscar y no ingresa nada se muestra: "Por favor, ingresa el número de empleado".
- Si el usuario ingresa letras, se muestra: "El formato del número de empleado es incorrecto".



La segunda pestaña solicita el número de empleado y se muestra el número de ordenes y el precio total de dichas ordenes realizadas por ese mesero.

De acuerdo a la información que se ingrese en el recuadro se pueden presentar las siguientes situaciones:

- Si el usuario ingresa un número y el empleado es mesero, se muestra su número de empleado, el número de ordenes y el precio total de dichas ordenes.
- Si el usuario ingresa un número y el empleado no es mesero se muestra: "El empleado no es mesero o no existe", para ello podemos auxiliarnos de la primer pestaña y confirmar que el empleado no es mesero.
- Si el usuario ingresa un número y el empleado no existe se muestra: "El empleado no es mesero o no existe", para ello podemos auxiliarnos de la primer pestaña y verificar la existencia del empleado.
- Si el usuario ingresa letras, se muestra: "El formato del número de empleado es incorrecto".
- Si el usuario oprime el botón buscar y no ingresa nada se muestra: "Por favor, ingresa el número de empleado".



La tercera pestaña solicita una fecha y se muestra la cantidad de ordenes, la cantidad de ventas y el monto total de las ventas en la fecha ingresada. Es importante mencionar que la fecha puede ser ingresada empezando por el año o empezando por el día, ambas notaciones funcionan.

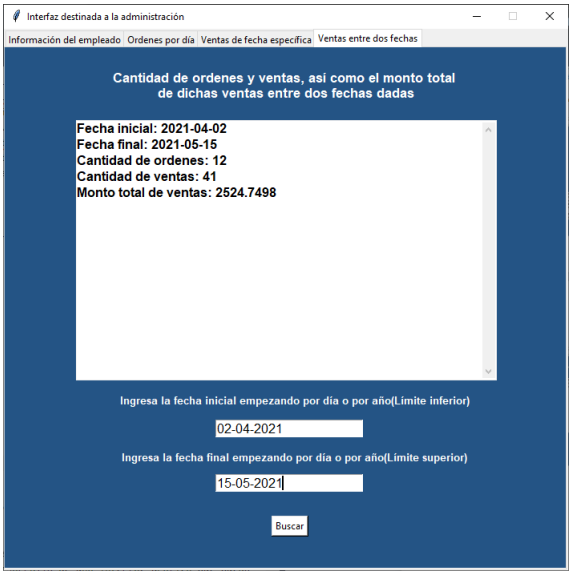
De acuerdo a la información que se ingrese en el recuadro se pueden presentar las siguientes situaciones:

- Si el usuario ingresa una fecha y existen ordenes y ventas asociadas a esa fecha, se muestra también el monto total de las ventas.
- Si el usuario ingresa una fecha y no se encuentran ventas se muestra: "No se encontraron ventas en esa fecha'.
- Si el usuario ingresa la fecha sin estar separada por guiones o diagonales, Ingresa letras o no sigue el formato empezando por el año o por el día se muestra:"El formato de fecha es incorrecto".
- Si el usuario oprime el botón buscar y no ingresa nada se muestra: "Por favor, ingresa una fecha".

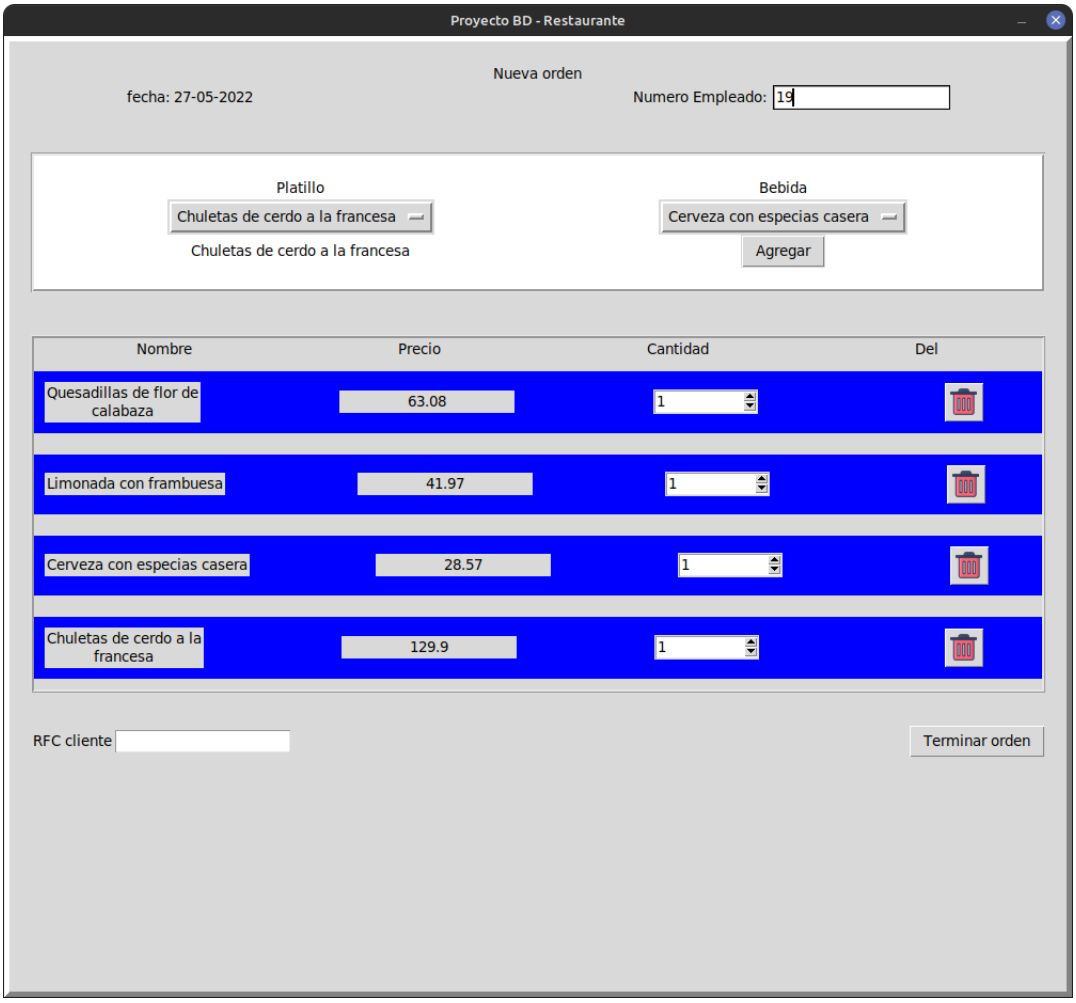
La cuarta pestaña solicita dos fechas y se muestra la cantidad de ordenes, la cantidad de ventas y el monto total de las ventas entre las fechas ingresadas. Es importante mencionar que la fechas pueden ser ingresadas empezando por el año o empezando por el día, ambas notaciones funcionan.

De acuerdo con las fechas que se ingresen en los recuadros se pueden presentar las siguientes situaciones:

- Si el usuario ingresa las fechas y existen ordenes y ventas en ese periodo de tiempo, se muestra también el monto total de las ventas.
- Si el usuario ingresa las fechas y no existen ventas en ese periodo de tiempo, se muestra: "No se encontraron ventas entre esas fechas'.
- Si el usuario no ingresa ninguna fecha, o no ingresa alguna de ellas , se muestra:"Por favor, ingresa las fechas".
- Si el usuario ingresa la fechas sin guiones o diagonales, ingresa letras o no sigue el formato empezando por el año o por el día se muestra:"El formato de las fechas es incorrecto, revísalo e intenta de nuevo".
- Si el usuario oprime el botón buscar y no ingresa nada se muestra: "Por favor, ingresa una fecha".



Para la parte de la interfaz destinada al ingreso de ordenes, el reto mas desafiante que tuvimos fue el manejo dinámico al añadir y quitar productos de una orden y al aumentar o disminuir su cantidad, sin embargo este problema fue solucionado con el uso de tkinter junto a los eventos que los elementos de la librería proporcionan. A continuación se muestra una captura de la interfaz:



Como podemos observar la interfaz es muy intuitiva, tiene dos menús para seleccionar un platillo o una bebida y un botón para agregar dicho producto a la orden. Esta interfaz se creo con el concepto 'CRUD', ya que en la parte de las ordenes podemos crear un producto que va en la orden, podemos visualizar cada producto y su información importante como el precio y la cantidad, podemos actualizar la cantidad de cada producto y por ultimo si el cliente lo desea podemos remover un producto antes de realizar la orden.

De igual forma, podemos observar que tenemos dos campos para ingresar el numero del empleado(mesero) y el RFC del cliente. El primer campo es necesario para realizar una orden pues siempre debe estar vinculada a un mesero y el RFC del cliente es un campo opcional pues este campo se rellena solamente cuando el cliente vaya a requerir de factura para su orden.

8. Conclusiones

Andrés Urbano Andrea
Con el desarrollo del presente proyecto pude elaborar, comprender y analizar el desarrollo de una base de datos desde su fase inicial con el desarrollo conceptual hasta su fase final con el desarrollo de una interfaz gráfica que pueda interactuar con ella. Como sucede en un proyecto real de implementación de una base de datos, ideamos un plan de trabajo para organizar las actividades que desarrollaría cada integrante del equipo. Utilizamos los conocimientos adquiridos durante el curso de base de datos para poder desarrollar la base de datos de forma adecuada en cada una de las fases de diseño. Por una parte, en el diseño conceptual de la base de datos, analizamos los requerimientos del problema y lo representamos por medio de un modelo entidad-relación extendido (MERE). Por otra parte, en el diseño lógico representamos el MERE en tablas. Realizamos la normalización de cada una de estas tablas para representar de manera óptima la información en cada tabla. En el diseño físico, aplicamos las sentencias básicas del DDL para creación de las tablas y estructuras de la base de datos. Así como comandos del DML para la modificación, actualización y eliminación de las tablas. Nos dimos la tarea de crear registros que contuvieran información veraz sobre los datos que puede almacenar un restaurante. Comprendí la gran utilidad que tienen las funciones, disparadores y vistas para agregarle funcionalidad a la base de datos. Por una parte, creamos funciones que pueden calcular atributos, como calcular la cantidad de ordenes y ventas en un periodo. Creamos disparadores que implementan restricciones de tipo check y que se ejecutan en un momento determinado, como el disparador que actualiza la cantidad vendida en una orden, cuando se incluye un nuevo producto en la orden. Por otra parte, las vistas nos ayudaron a desplegar cierta información al usuario, tal como sucede con la factura. El desarrollo de la interfaz gráfica fue un reto, ya que teníamos que tener conocimientos del uso de las librerías Psycopg2 y Tkinter en Python. Y aunque, no fue posible almacenar la fotografía del empleado en la base de datos, ideamos una solución para mostrarlas a partir de un URL. Logramos conectar la base de datos con la interfaz gráfica y el resultado se asemeja bastante a un sistema real de base de datos de un restaurante.

Calvillo Martínez Erik Jonathan
Gracias a este proyecto pude darme cuenta de la gran carga de trabajo que implica el diseñar una base de datos. Es por ello que es importante implementar la teoría vista a lo largo del curso para asegurarnos de realizar un buen trabajo.

El diseño de cualquier base de datos puede resumirse en las siguientes fases o etapas:

1. Obtención del MER o MERE según sea el caso.
2. Obtención del MR en su representación intermedia.
3. Normalizar hasta la 3FN.
4. Obtención del MR en su representación final después de normalizar.
5. Generación del código DDL asociado a nuestra base de datos.

Nosotros como equipo tratamos de seguir todas estas etapas de manera conjunta, sin embargo, por cuestiones de tiempo decidimos dividirnos las tareas para lograr concluir este proyecto. Desde mi punto de vista puedo decir que realizar este proyecto fue bastante difícil, pues los conocimientos que implica realizarlo no se consiguen de la noche a la mañana. La comunicación entre el equipo fue clave para lograr culminar lo solicitado, a pesar de que mis compañeras Arely y Karla se encargaban del DML siempre se comunicaban con nosotros, pues como todos los integrantes tenemos conocimientos sobre la materia, podíamos encontrar una solución rápidamente mientras seguíamos trabajando de manera paralela. Me alegra que parte de la interfaz me haya sido asignada a mi, porque en el semestre del paro decidí tomarme el tiempo de aprender a realizar interfaces gráficas pues sabía que en algún momento iba a necesitar de ellas.

El diseño de la interfaz gráfica fue todo un reto, porque si bien tenia conocimientos básicos de tkinter, jamás había realizado una interfaz enlazada a postgreSQL, el adaptador psycopg2 fue de gran ayuda, pues me permitió enlazar a mi interfaz con la base de datos sin ningún problema, las instrucciones que implementa son muy fáciles de inferir siempre y cuando tengas conocimientos básicos de Python y de lenguaje SQL. Un problema bastante grande de la interfaz fue el lograr visualizar la fotografía del empleado dentro

de ella. Durante las primeras versiones la fotografía se obtenía directamente de una ruta ubicada de manera local en mi computadora, si este detalle no se corregía, este punto iba a estar mal, pues se solicita que la imagen se obtenga directamente de la base de datos. Como el equipo tuvo problemas para subir imágenes a la base datos optamos por únicamente guardar dentro de ella el URL asociado a la foto de cada empleado, esta solución es bastante válida, pues si nos ponemos a pensar a futuro, el tamaño de una base de datos con millones de registros asociados a los empleados puede crecer exponencialmente, el guardar las fotos en internet y obtenerlas de ahí vuelve mas eficiente nuestra base de datos.

Inicialmente tenía la idea de realizar 4 interfaces, cada una de ellas con propósito en particular, sin embargo, me di cuenta que esto no era lo más conveniente, es por ello que decidí realizar una sola interfaz asociada a la parte administrativa del restaurante, esta interfaz se divide en 4 pestañas, cada una de ellas realiza una función distinta, según se considere.

- La primer pestaña fue la más sencilla debido a que esta interactúa de manera directa con la tabla empleado logrando mostrar toda su información y su fotografía.
- La segunda pestaña muestra el número de empleado, el número de ordenes y el precio total de las ordenes asociadas a ese empleado siempre y cuando el empleado sea mesero.
- La tercera pestaña muestra la cantidad de ordenes, la cantidad de ventas y el monto total asociado a una fecha dada.
- La cuarta pestaña funciona de manera similar a la anterior con la diferencia de que esta necesita de dos fechas.

Para la implementación de la pestaña dos en adelante la comunicación con mi compañera Karla fue de vital importancia, pues ella debía tener conocimiento de como funcionaba la librería psycopg2 es por ello que nos reunimos más de una ocasión para llegar a una solución en conjunto, en nuestras reuniones de zoom yo le explicaba que necesitaba para que la interfaz me arrojara la respuesta esperada y entre los dos lográbamos plantear alguna idea que nos permitiera llevar esta etapa de la mejor forma posible.

Para finalizar puedo decir que este proyecto fue un gran reto para mi formación como ingeniero, pues en este tipo de trabajos es donde realmente te das cuenta que tanto has aprendido en la carrera, porque si bien se necesitan conocimientos de bases de datos, también se necesitan conocimientos de otras materias como EDA o POO que si bien no están ligadas de manera directa, su estudio nos permitió mejorar nuestra lógica de programación, la cual fue aplicada en las funciones, en los triggres y la interfaz, he de aclarar que cuando llegué a la facultad yo tenía pensado dedicarme a la programación de apps móviles, o videojuegos, pero el dedicarme 100 % a las bases de datos no es mala idea, o mejor aún podría enfocarme en ambas cosas para lograr crear buenos proyectos.

De La Cruz Munguia Arely
Al realizar este proyecto me enfrente a grandes retos, uno de ellos fue trabajar en equipo ya que dividir las tareas es un reto difícil pues cada uno de los integrantes debe tener muy claro cuales son sus habilidades y así poder hacer uso de estas dentro del desarrollo del trabajo. Considero que la organización en un principio fue un tanto difícil pues a mi parecer consideramos que teníamos tiempo de sobra e iniciamos un poco tarde con todo y las reuniones con todo el equipo eran escasas debido a la agenda que cada uno tenia. La elaboración del Modelo Entidad Relación y el mapeo intermedio al Modelo Relacional lo realizamos en conjunto, aunque a lo largo de la implementación y de varias cuestiones e ideas que surgieron durante el proceso se fue modificando y esto en parte nos quitaba tiempo por que llegábamos a confundirnos en ocasiones, debíamos consultar cada uno de los cambios y ajustes que se pensaban hacer con los miembros del equipo pues así todos lográbamos ver que podría mejorar o empeorar esta nueva idea de cambio.

Una vez que logramos tener los ajustes adecuados continuamos con el siguiente paso del diseño que era la normalización, que al igual que el MER y MR se hizo con la participación de todo el equipo, el cuál fue un proceso a mi parecer muy importante de realizar y analizar a detalle pues si no lo hacíamos bien podríamos llegar a un resultado que complicaría ciertos procesos que se llevarían a cabo con la manipulación

e interacción de los registros que llevaría la base de datos. Cuando se hizo este proceso inicialmente se decidió ignorar ciertas dependencias transitivas en la relación "EMPLEADO" pues tendríamos muchas tablas con la información separada y para los fines que serian ocupadas no nos convenía separar de esa forma tan especifica la información. Fue una decisión importante pero que se tomo con la aprobación de todo el equipo.

Una vez que ya se tenia todo esto y viendo que se tenían ciertas dificultades para poder reunirnos todos y que trabajar todos en una sola actividad nos retrasaría más, optamos por continuar con el trabajo dividiéndolo en tareas especificas para cada integrante, de esta forma lo haríamos más rápido y terminaríamos a tiempo. Junto con mi compañera karla realizamos la implementación del DDL, y la verdad es que fue un proceso bastante complicado y pesado por la parte de la implementación de funciones, triggers, disparadores y considerar todas las características que debían tener cada una de estas respetando la integridad y la solución propuesta.

Para la creación del código de las tablas hicimos uso de la herramienta ER Studio que nos permitía generar el modelo físico (únicamente las tablas con sus respectivas restricciones) y así nosotros tuvimos una idea más general de como escribir de la mejor forma posible las sentencias para la creación de nuestra base de datos. En cuanto a las restricciones de llaves foráneas tuvimos que decidir que tipo de restricción debía tener para la actualización y borrado de información y para esto se plantearon situaciones ficticias que ayudaran a entender de mejor manera cuales eran nuestras mejores opciones, pero si fue bastante complejo llegar a un acuerdo. Otro de los retos fue lograr hacer la función que nos permitiera generar el folio de la orden al momento de insertar registros dentro de esta tabla, nos tomo mucho tiempo pero al final logramos encontrar una solución que si bien no sabemos es la más óptima resuelve el requerimiento, y no sabemos si es óptima pues hicimos uso de un secuenciador que no siempre es lo más recomendable de usar dentro de una llave primaria.

En cuanto a los disparadores fueron de gran ayuda para poder realizar ciertas operaciones en momentos determinados mientras se manipula a la base de datos, estos nos ayudan a actualizar datos como la cantidad vendida de los productos, verificar que efectivamente el empleado que esta ingresando la orden sea un mesero, corroborar si un producto sigue estando disponible, hacer el calculo del precio total de la orden, al igual que el precio total por tipo de producto, también se calcula la cantidad total de productos que incluye una orden y se calcula al edad del empleado a partir de la fecha de nacimiento. Todos estos son funcionalidades de la base de datos que hacen uso de atributos que ya se tienen para poder obtener otros. Todo esto fue un reto debido a que los conocimientos sobre estos eran mínimos debido a que no se vieron con tanta profundidad en teoría y el laboratorio.
En cuanto a la parte de funciones se implementaron algunas como la que calcula el folio de la orden, y otras que se tuvieron que hacer 2 veces debido a que dentro de la base de datos funcionaban bien pero para la interfaz no lográbamos que funcionaran correctamente debido al poco tiempo que tuvimos para investigar estos detalles que se generaban si usábamos ciertas sentencias como RAISE EXCEPTION dentro de python.
Finalmente optamos por crear un índice semejante a un NON CLUSTERED dentro de la columna nombre en la tabla producto ya que este atributo seria consultado varias veces, su implementación fue sencilla y creo que si mejoro el tiempo de búsqueda. Añadiendo a esto otro reto que nos enfrentamos fue la creación del formato de la factura, en primer instancia habíamos creado una vista con 4 inner join con 4 tablas distintas, pero ocurría el problema de que la información del cliente salia repetida si había más de un producto de la orden, habíamos optado por dejarlo así pero al final se encontró una solución gracias al uso de una función y cursores, todo esto con la ayuda de mi compañera andrea.
Por parte de la interfaz gráfica se hizo uso de Python ya que no se tenia un amplio conocimiento para poder implementar algo más complejo y se logro obtener algo que mostraba de la mejor manera las funcionalidades necesarias y requeridas.
Sin duda fue un proyecto bastante demandante y complejo pero que nos ayudo a aplicar todos los conceptos vistos durante todo el semestre, admito que fue un reto muy grande y más por el tiempo y las complicaciones que se presentaron y que sin duda no lo habríamos logrado si no nos hubiéramos repartido el trabajo. Realmente considero que es de gran importancia saber trabajar en equipo, organizar los tiempos y tener bastantes claros cada uno de los conceptos así como los requerimientos que se piden, en proyectos así se debe interpretar de la mejor manera y siempre preguntar al cliente que es lo que desea obtener pues podemos llegar a malinterpretar ciertas cosas y si ya se tiene un gran avance podría perderse mucho tiempo corrigiendo

todo. Por eso es importante tener todo de la mejor manera posible en todas las fases del diseño de una base de datos.
Considero que aprendí mucho pero sin duda debo repasar mucho más ya que estos temas realmente son muy aplicables a la vida real y podrían ayudarme bastante en mi formación académica y profesional saberlos utilizar e implementar de la mejor manera posible y gracias a Postgres siento que pude entender mejor con ayuda de la interfaz gráfica de PgAdmin.

Vázquez Pérez Karla

Al realizar este proyecto me he enfrentado a dificultades de organización en tiempos, así como he presentado dificultades al intentar plasmar ideas en herramientas de PostgreSQL aún no vistas en teoría y muy poco vistas en laboratorio, como es el uso de disparadores y funciones. Admito que al principio nos fue un poco difícil entre el equipo ponernos en acuerdo de cada parte de los modelos planteados. Debíamos estar los cinco en acuerdo para poder seguir avanzando, por lo que nos atrasamos un poco. De igual manera los horarios de cada integrante para poder reunirnos era difícil acomodarlos, por lo que optamos por repartirnos el trabajo una vez que el Modelo Entidad Relación y el Modelo Relacional estaban listos. Cabe aclarar que para normalizar optamos por no usar las dependencias transitivas de rfc con estado, fecha de nacimiento y nombre completo de empleado, ya que de esta manera no considerábamos que nos fuera a quedar de una manera práctica la información de los empleados. De igual manera optamos por no usar la dependencia del código postal con la colonia, ya que concluimos que tampoco era necesario.

En mi caso personal y como se ha puesto en el documento, me ha tocado realizar una parte del archivo SQL. Mi mayor reto fue aprender a realizar disparadores y funciones cuando en mis clases de laboratorio no fueron muy bien explicados estos temas. Investigué en varias fuentes ejemplos, teoría y tips que me podrían ser de ayuda para realizarlos y poder cumplir con varios de los requerimientos del proyecto, como es el cálculo automático de valores de diferentes atributos, triggers para mostrarnos errores y funciones para mostrarnos consultas a través de la interfaz.

Poco a poco conforme íbamos avanzando en el código SQL, nos teníamos que ir a regresar a nuestros modelos de entidad relación y al relacional para cambiarles ciertos detalles y que se adaptaran a nuestra base de datos. Por ejemplo, el atributo cantidad_vendida no estaba en un principio pero se decidió colocarlo para facilitar la vista del platillo más vendido.

Para la creación de las tablas discutimos en equipo las restricciones de borrado y actualizado que debíamos implementar en las tablas hijas y padre, y en varias ocasiones tuvimos que debatir si era buena idea dejar como participación opcional a varias relaciones, como fue en el caso de la tabla hija Orden con la tabla Empleado y Cliente, ya que decidimos ahí implementar restricción de SET NULL en el borrado para no perder información de una orden si se desea eliminar a un cliente o empleado de la base de datos. Para ellos debíamos hacer a las llaves foráneas en Orden como NULL. Lo mismo pasó con la tabla hija Producto con la tabla padre Categoría, en la cual decidimos colocar un SET NULL en la restricción de borrado.

Varias veces se tuvieron que corregir las funciones, vistas y disparadores cuando ya se creían listos debido a errores que conforme probamos la base de datos iban surgiendo, como es el caso de las funciones mesero_ordenes_al_dia_bdd_function, mesero_ordenes_al_dia_function, cantidad_monto_ventas_fecha_bdd_function, cantidad_monto_ventas_fecha_function, cantidad_monto_ventas_fechas_bdd_function y cantidad_monto_ventas_fechas_function.

Estas 6 funciones tuvieron unos detalles y el principal fue que mediante nuestra interfaz de Python, no se podían hacer notar los errores llamados por RAISE EXCEPTION, por lo que optamos por hacer dos funciones para cada una. Es decir, tenemos mesero_ordenes_al_dia_bdd_function() y mesero_ordenes_al_dia_function(). El primero funciona mediante RAISE EXCEPTION para indicar que el numero_empleado dado no es mesero o no existe ese empleado en la base de datos, mientras que el segundo devuelve una tabla vacía si ese es el caso y mediante Python se ha agregado condición de que si regresa una tabla vacía, le imprima al usuario que el empleado ingresado no es mesero o no existe. Lo mismo para las funciones de dar una fecha o dos fechas dadas.

Para los triggers de precio_total_trigger, cantidad_vendida_trigger y cantidad_total_trigger tuvimos que modificarlas para darle la posibilidad al cliente de borrar algún producto de su orden y que este producto ya no se tome en cuenta para su cantidad_total, precio_total ni para la cantidad_vendida de dicho producto.

Nos dimos cuenta que necesitábamos un trigger que marcara error si el numero_empleado que se intente asociar a una orden no fuera un mesero o no existiera (en este último caso, por ser llave foránea siempre va a mostrar error de todas maneras). Para ello implementamos el uso del trigger mesero_a_orden_trigger.

Otras dificultades a las que nos enfrentamos fue el uso de fotos para los empleados. No logramos implementar bien la codificación y decodificación de las fotos por postgreSQL ni por la interfaz utilizada, por lo que optamos por el uso de links que nos llevan a la foto y que automáticamente se muestran en nuestra interfaz.

Sin lugar a duda dos de nuestras dificultades más grandes fueron la creación de la factura y el cálculo automático de los folios de las órdenes. Para este último optamos por el uso de un secuenciador y una función, el secuenciador dándonos los valores secuenciales del folio y la función haciendo una concatenación de las letras "ORD-" junto con el casteo de lo que nos estuviera arrojando el secuenciador. Resolver la factura fue lo que nos ha tomado más tiempo en la base de datos y esto se debía a no poder deshacernos de los registros repetidos de las órdenes que tuvieran más de un producto diferente. Esto lo arreglamos con una función que arroja mensajes mediante RAISE NOTICE en lugar de que nos devuelva la información en una tabla.

Puedo concluir que ha sido un proyecto en el cual he trabajado y aprendido sobre los temas del curso de una manera muy buena, pero demandante. Al hacer una base de datos se deben tomar en cuenta muchos aspectos que quizá no se ven a simple vista en el diseño de los primeros modelos, y que conforme vas creándola, te vas dando cuenta de errores que se han cometido en las etapas de diseño anteriores. Se deben tomar decisiones que podrían alterar nuestra base de datos de una cierta manera para beneficiarla de otra, sin embargo, son importantes para cumplir ciertos requerimientos que se den. Sin duda este proyecto me ha ayudado a entender mejor cómo se debe crear una buena base de datos y que tantas dificultades puedes enfrentarte al hacerla para poder cumplir con lo solicitado y además no dejar de lado puntos de seguridad y rendimiento de nuestras base de datos.

Vázquez Sánchez Erick Alejandro

La realización de este proyecto me ayudó a cimentar todos los conocimientos aprendidos dentro de la teoría y la práctica de la materia. Además de que me ayudó a comprender la utilidad e importancia de las bases de datos en el desarrollo de un proyecto de software, pues recuerdo que en materias anteriores, como lo eran EDA y POO, era todo un problema realizar proyectos sin poder hacer que los datos modificados se guardaran para la siguiente ejecución del programa.

Por otra parte, este proyecto me ayudó a darme cuenta de la importancia de la definición de un plan para llegar a un objetivo en un equipo de trabajo, pues aunque tuvimos bastante tiempo para establecer este plan y terminar sin ninguna complicación el proyecto, al final sentí que el tiempo se nos venia encima. De la mano de esto puedo mencionar también que realizamos reuniones para discutir cuestiones sobre el proyecto, en las cuales por diferencia de opiniones no se avanzaba demasiado, pero creo que estas diferencias de opiniones de cierta manera nos ayudaron a observar y analizar de mejor forma los requerimientos del proyecto.

Por ultimo, lo que mas se me dificulto fue la realización de la interfaz gráfica, ya que el tiempo se me vino encima ya que tenia que realizar entregas de trabajos de otras materias las cuales hicieron que me retrasara un poco conforme al avance de mis compañeros con su parte.

9. Fuentes consultadas

[1] León, A., Ortiz, Y. (2015). *PL/pgSQL y Otros lenguajes procedurales en PostgreSQL*. La Habana, Cuba. Universidad de las Ciencias Informáticas. Primera edición. Creative commons.

[2] Chávez, J. (2019). *Fundamentos de programación en lenguaje PL/pgSQL*. Venezuela. Universidad Politécnica Territorial de Aragua. IEASS, Editores.

[3] The PostgreSQL Global Development Group. (2022). *Triggers*. Documentation PostgreSQL 14.[Online]. Disponible en: https://www.postgresql.org/docs/14/triggers.html

[4] The PostgreSQL Global Development Group. (2022). *PL/pgSQL: SQL Procedural Language*. Documentation PostgreSQL 14.[Online]. Disponible en: https://www.postgresql.org/docs/14/plpgsql.html

[5] Python Software Foundation.(27 de mayo de 2022). *Interfaces gráficas de usuario con Tk (Tkinter)*. Documentación.[Online]. Disponible en: https://docs.python.org/es/3/library/tk.html

[6] The Psycopg Team. (s.f.). Psycopg 2.9.3 documentation. Documentación. [Online]. Disponible en: https://www.psycopg.org/docs/

[7] Phyton documentation. (s.f.). Cómo obtener recursos de Internet con el paquete urllib. [Online]. Disponible en:
https://docs.python.org/es/3/howto/urllib2.html