



Universidad Autónoma de México
FACULTAD DE INGENIERÍA

PROYECTO FINAL BASES DE DATOS

Profesor:
Fernando Arreola Franco

Autores:

Borboa Castillo Carlos Alfonso
Castillo Montes Pamela
Espinosa Cortez Giselle
Galán Olivares Miguel Angel
Ortíz Rivera Miguel Angel

Mayo 2022

Índice

1. Introducción	2
2. Plan de Trabajo	5
3. Diseño	7
3.1. Identificación de Entidades y Atributos	7
3.2. Diagrama DER	10
3.3. Representación Intermedia y MR	11
3.4. Normalización	14
4. Implementación	21
4.1. Creación de tablas	21
4.2. Campos Calculados (Triggers y Funciones)	25
4.2.1. Secuencias	25
4.2.2. Fecha de nacimiento y Edad	25
4.2.3. Generar Ordenes	26
4.2.4. Agregar Productos	27
4.2.5. Cantidad ordenes por mesero	28
4.2.6. Bebida y Platillo más vendido	29
4.3. Inserción de datos en el manejador	32
4.3.1. Archivos .csv	32
4.3.2. Mediante script	35
4.4. Índice	37
5. Presentación	37
5.1. Interfaz Gráfica	37
5.1.1. Conexión con la BD	37
5.1.2. Interfaz	42
6. Conclusiones	47

Objetivo

Se lograra automatizar la taqueria "Los inges", controlando los empleados que existen y si se desean agregar más, las ordenes que estan en proceso, las ordenes concluidas, los productos disponibles, el contenido de una orden, la facturación, entre otras cosas por medio de claves que nos permitan acceder a la información requerida.

1. Introducción

En la parte correspondiente a diseño, comenzamos haciendo el analisis del problema leimos detallamente cada uno de los renglones identificando las entidades mínimas para que el negocio pudiese funcionar y los atributos que corresponden a cada una de ellas.

A partir de este analisis obtuvimos el MER, de manera general este cuenta con 10 entidades, entre ellas existen relaciones (1,M) y (M,M)

Se identifico que la entidad empleado es un supertipo, mientras que mesero, cocinero y administrativo son subtipos de este, ya que compartian atributos en común, pero entre ellos tenian atributos que solo tenian sentido para una subtipo y para los otros no decimos utilizar especialización, ademas uno de los requerimientos es que el empleado puede tener mas de un puesto por lo que utilizamos una relación parcial.

Otra entidad que se puede considerar necesito un con 'tratamiento especial' es 'dependiente', la cual evidentemente no puede existir sin estar asociada a un empleado por lo que es se le considera como una entidad debil.

Continuamos con la entidad producto que se identifico como supertipo y se derivan los subtipos bebida y platillo donde tambien se aplico una especialización, pero esta vez con una relacion parcial, ya que un producto solo puede ser bebida o platillo. Aplicamos tambien un atributo discriminante como bandera para identificar de que subtipo se trata.

Las otras entidades no emplean ningun concepto de Modelo EntidadRelacional Extendido.

Pasamos a la representacion intermedia donde ya no solo nos importa la existencia de nuestros datos sino tambien la representación de ellos. El concepto más importante en esta parte es el de llave foranea, ya que a partir de esta podremos acceder a otras tablas

Para el mapeo hay un atributo multivaluado que es telefono por lo que se crea la relación según las reglas correspondientes, ademas tenemos una relación (M,M) donde tambien se crea una nueva relación. A partir de la representación intermedia se pudieron mapear de manera exitosa las entidades y llegar al modelo relacional. Aunque el equipo opto por hacer las tablas desde la representacion intermedia y no esperando a realizar el MER, se tuvo que esperar a que la base de datos estuviera terminada para asi poder incluir el MR con ingenieria inversa.

Al aplicar la normalización se crearon varias nuevas relaciones que nos permiten conservar la integridad de nuestra informacion por lo que decidimos realizar nuevamente el modelo relacional para incluirlas. Las consideramos como catalogos

A pesar de solo basarnos en la representación intermedia para realizar las tablas en el manejador logramos implementarlas de manera exitosa.

Siempre debemos cuidar la integridad relacional de nuestras tablas, como es el caso de catalogo_estados y empleado, no puedo referenciar algo que aún no existe debo crear primero catalogo_estados y agregar información para despues poder referenciarla en empleado, tambien es sumamente importante al crear una tabla la implementacion de los CONSTRAINTS, los atributos que son calculados como en este caso la edad y la fecha de nacimiento que podian obtenerse a partir del rfc del empleado, los CHECKS, las secuencias, las funciones y los triggers.

Para este proyecto se crearon dos secuencias cuenta_num.emp y genera_Folio (este se concatena con ORD) y se crearon las funciones como: La función que nos permite obtener la fecha de nacimiento y la edad, seguido del trigger que se ejecuta despues de aplicar una sentencia DDM y actualiza los campos. Se hace una función parecida para cliente, pero al no contener edad no podemos usar la mencionada anteriormente. Tambien se crean funciones

para agregar ordenes, agregar productos a las ordenes registrada, la cantidad de ordenes que tiene un mesero, el platillo mas vendido, la bebida más vendida son que se consideraron necesarias para el funcionamiento correcto.

Es muy importante saber distinguir si se trata de un objeto para asi ocupar instrucciones DDL o si quiero operar interactuar con los datos para asi ocupar sentencias de tipo DDM.

Decidimos hacer la inserción de los datos tanto por archivos .csv y con la setencia de DML INSERT, ambas son maneras efectivas de insertar la información en nuestras tablas.Para nuestro caso en la inserción era sumamente importante saber el orden de mis atributos en mis tablas.

Para la interfaz gráfica se hizo uso de Visual Basic, la parte medular de esta sección es conectar la base de datos con el proyecto en Visual. Se implementó una estructura cliente- servidor, esto quiere decir que una computadora tiene la base de datos y trabaja como servidor, mientras qué hay otra donde se encuentra la interfaz.

Esto hace que el cliente pueda hacer peticiones a la computadora servidor.

Hablando únicamente de la interfaz esta cuenta con 4 secciones:Nuevo Cliente, Agregar productos, cerrar/factura y consultas que se habilitan con un botón.

Las consultas abarcan la cantidad de órdenes atendidas por un mesero, platillo y bebida más vendido, productos disponibles y ventas realizadas en un periodo de tiempo y la opción de ver todos los empleados que trabajan en un restaurante

]El equipo opto porque los requerimientos propuestos por el profesor corresponderian a una taqueria la nombramos "los inges".

En la parte correspondiente a diseño, comenzamos haciendo el analisis del problema leimos detallamente cada uno de los renglones identificando las entidades mínimas para que el negocio pudiese funcionar y los atributos que corresponden a cada una de ellas.

A partir de este analisis obtuvimos el MER, de manera general este cuenta con 10 entidades, entre ellas existen relaciones (1,M) y (M,M)

Se identifico que la entidad empleado es un supertipo, mientras que mesero, cocinero y administrativo son subtipos de este, ya que compartian atributos en común, pero entre ellos tenian atributos que solo tenian sentido para una subtipo y para los otros no decimos utilizar especialización, ademas uno de los requerimientos es que el empleado puede tener mas de un puesto por lo que utilizamos una relación parcial.

Otra entidad que se puede considerar necesito un con 'tratamiento especial' es 'dependiente', la cual evidentemente no puede existir sin estar asociada a un empleado por lo que es se le considera como una entidad debil.

Continuamos con la entidad producto que se identifico como supertipo y se derivan los subtipos bebida y platillo donde tambien se aplico una especialización, pero esta vez con una relacion parcial, ya que un producto solo puede ser bebida o platillo.Aplicamos tambien un atributo discriminante como bandera para identificar de que subtipo se trata.

Las otras entidades no emplean ningun concepto de Modelo EntidadRelacional Extendido.

Pasamos a la representacion intermedia donde ya no solo nos importa la existencia de nuestros datos sino tambien la representación de ellos. El concepto más importante en esta parte es el de llave foranea, ya que a partir de esta podremos acceder a otras tablas

Para el mapeo hay un atributo multivaluado que es telefono por lo que se crea la relación según las reglas correspondientes, ademas tenemos una relación (M,M) donde tambien se crea una nueva relación. A partir de la representación intermedia se pudieron mapear de manera exitosa las entidades y llegar al modelo relacional. Aunque el equipo opto por hacer las tablas desde la representacion intermedia y no esperando a realizar el MER, se tuvo que esperar a que la base de datos estuviera terminada para asi poder incluir el MR con ingenieria inversa.

Al aplicar la normalización se crearon varias nuevas relaciones que nos permiten conservar la integridad de nuestra información por lo que decidimos realizar nuevamente el modelo relacional para incluirlas. Las consideramos como catalogos

A pesar de solo basarnos en la representación intermedia para realizar las tablas en el manejador logramos implementarlas de manera exitosa.

Siempre debemos cuidar la integridad relacional de nuestras tablas, como es el caso de catalogo_estados y empleado, no puedo referenciar algo que aún no existe debo crear primero catalogo_estados y agregar información para despues poder referenciarla en empleado, también es sumamente importante al crear una tabla la implementacion de los CONSTRAINTS, los atributos que son calculados como en este caso la edad y la fecha de nacimiento que podian obtenerse a partir del rfc del empleado, los CHECKS, las secuencias, las funciones y los triggers.

Para este proyecto se crearon dos secuencias cuenta_num_emp y genera_Folio (este se concatena con ORD) y se crearon las funciones como: La función que nos permite obtener la fecha de nacimiento y la edad, seguido del trigger que se ejecuta después de aplicar una sentencia DDM y actualiza los campos. Se hace una función parecida para cliente, pero al no contener edad no podemos usar la mencionada anteriormente. Tambien se crean funciones para agregar ordenes, agregar productos a las ordenes registrada, la cantidad de ordenes que tiene un mesero, el platillo mas vendido, la bebida más vendida son que se consideraron necesarias para el funcionamiento correcto.

Es muy importante saber distinguir si se trata de un objeto para asi ocupar instrucciones DDL o si quiero operar interactuar con los datos para asi ocupar sentencias de tipo DDM.

Decidimos hacer la inserción de los datos tanto por archivos .csv y con la setencia de DML INSERT, ambas son maneras efectivas de insertar la información en nuestras tablas.Para nuestro caso en la inserción era sumamente importante saber el orden de mis atributos en mis tablas.

Para la interfaz gráfica se hizo uso de Visual Basic, la parte medular de esta sección es conectar la base de datos con el proyecto en Visual. Se implementó una estructura cliente- servidor, esto quiere decir que una computadora tiene la base de datos y trabaja como servidor, mientras qué hay otra donde se encuentra la interfaz.

Esto hace que el cliente pueda hacer peticiones a la computadora servidor.

Hablando únicamente de la interfaz esta cuenta con 4 secciones:Nuevo Cliente, Agregar productos, cerrar/factura y consultas que se habilitan con un botón.

Las consultas abarcan la cantidad de órdenes atendidas por un mesero, platillo y bebida más vendido, productos disponibles y ventas realizadas en un periodo de tiempo y la opción de ver todos los empleados que trabajan en un restaurante

2. Plan de Trabajo

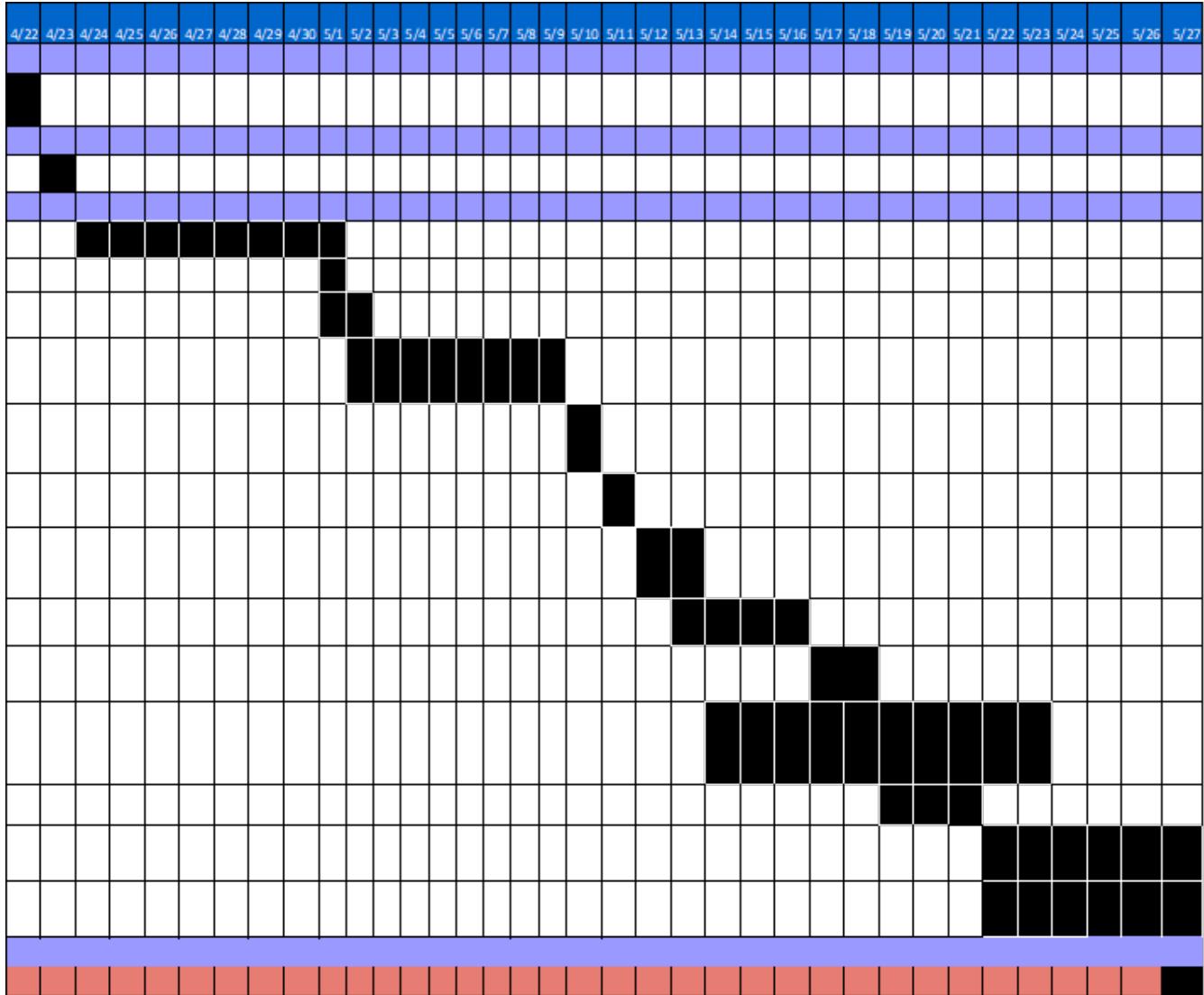


Figura 1: Parte del cronograma con las fechas distribuidas

Cuadro 1: Cronograma de Actividades

Tareas	Responsable	Fecha de inicio	Fecha final	Días	Estado
Planteamiento del Proyecto El profesor proporciona el documento con los requerimientos	Ing. Fernando Arreola Franco	22-abr	22-abr	1	Completado
Planificación del proyecto					
Reunión para delegar actividades	Todos los integrantes	23-abr	23-abr	1	Completado
Ejecución del Proyecto					
Diagrama Entidad-Relación	Miguel Galan Olivares	24-abr	01-may	7	Completado
Reunión para revisar DER	Todos los integrantes	01-may	01-may	1	Completado
Solicitar una reunión con el profesor para revisión	Miguel Ortiz Rivera	01-may	02-may	1	Completado
Representación Intermedia MR y Creación de tablas para el manejador	Miguel Ortiz Rivera y Carlos Borboa Castillo	02-may	10-may	8	Completado
Reunión para revisar representacion intermedia y tablas	Todos los integrantes	10-may	10-may	1	Completado
Solicitar una reunión con el profesor para revisión	Miguel Ortiz Rivera	11-may	11-may	1	Completado
Pasar tablas al manejador	Pamela Castillo Montes	11-may	13-may	2	Completado
Campos Calculados con Programación	Miguel Ortiz Rivera	13-may	16-may	3	Completado
Hacer los insert en las tablas	Giselle Espinosa Cortez	17-may	18-may	1	Completado
Inicia planteamiento y desarrollo de la interfaz gráfica	Carlos Borboa Castillo y Miguel Galan Olivares	14-may	23-may	9	Completado
Elaboración del MR con ingeniería Inversa	Espinosa Cortez Giselle	19-may	21-may	2	Completado
Elaboración del documento en LaTeX	Giselle Espinosa Cortez y Castillo Montes	22-may	27-may	5	Completado
Elaboración de la presentación	Giselle Espinosa, Carlos Borboa y Pamela Castillo	22-may	27-may	5	En curso
Entrega de Proyecto Final		27-may	27-may		En curso

3. Diseño

Para la primera parte del proyecto, la cual trata el tema relacionado al diseño de una base de datos, se pide que esta cumpla con los siguientes requisitos.

” Un restaurante desea digitalizar su forma de operación, para ello se desarrollará un sistema informatico 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**, **telefónos**, **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 solo 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 actualizar 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
- Dado un número de empleado, mostrar la cantidad de órdenes que han registrado en el día así como el total que se ha pagado por dichas órdenes.
- 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 asemejar 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
- El folio de la orden debe tener un formato similar a ORD-001, prefijo ORD, seguido de un guión y un número secuencial.
- Donde este presente el atributo **domicilio**, está compuesto por **estado**, **código postal**, **colonia**, **calle y número**.
- Donde este presente el atributo **nombre**, está compuesto por **nombre**, **apellido paterno y materno**. ”

3.1. Identificación de Entidades y Atributos

Analizando el texto anterior logramos identificar las siguientes entidades con sus respectivos atributos.

- **Empleado** de la cual identificamos los atributos: **Nombre (Compuesto)**, **número (Candidato)**, **RFC (Primario)**, **foto**, **sueldo**, **edad (Calculado)**, **teléfono (Multivalorado)**, **fecha de nacimiento (Calculado)**, **domicilio (Compuesto)**

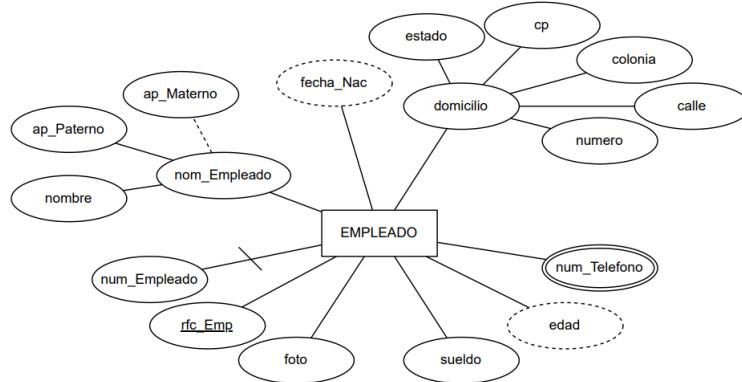


Figura 2: Entidad Empleado

En el caso de empleado, el problema nos indica que se tiene diferentes tipos de roles tales como "mesero, cocinero y administrativo" los cuales a su vez contienen diferentes atributos.

Al tratarse de una entidad que se separa en múltiples, procedemos a implementar una especialización teniendo en consideración una participación total, ya que se menciona que un empleado puede tener varios puestos, por lo tanto obtenemos la siguientes entidades.

- **Mesero** donde identificamos que su atributo característico es **horario**
- **Cocinero** donde identificamos que su atributo característico es **especialidad**
- **Administrativo** donde identificamos que su atributo característico es **rol**

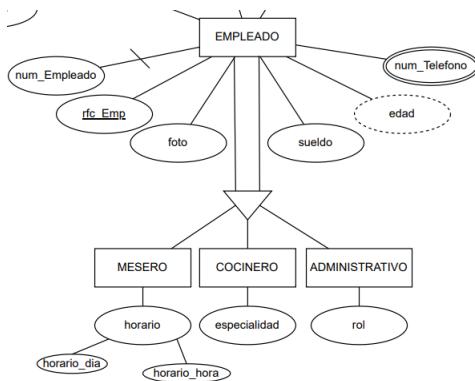


Figura 3: Especialización Empleado

Relacionado directamente a la entidad empleado tenemos una nueva llamada dependiente, la cual identificamos como entidad débil ya que su existencia depende de la de empleado, es decir, no se puede guardar registros de los dependientes si no se tiene un empleado al cual asociarlo.

Dado la anterior, relacionamos a la entidad empleado y dependiente usando el verbo descriptivo "tiene" y una cardinalidad 1:M, siendo la entidad dependiente.

- **Dependiente** de este identificamos los atributos: **Nombre (Compuesto)**, **curp (Primario)**, **parentesco**

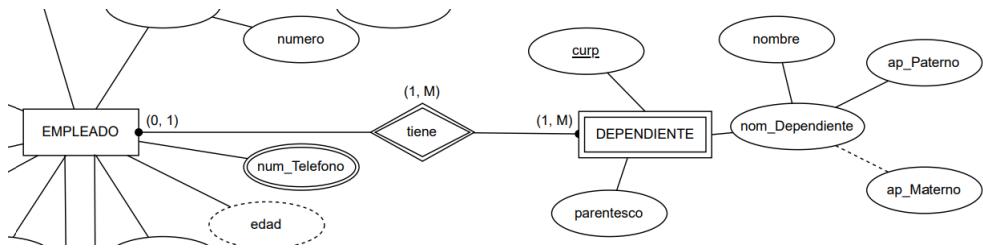


Figura 4: Entidad y relación de Dependiente

Por otro lado, tenemos la entidad producto

- **Producto** donde identificamos que tiene los atributos: **nombre** (**Candidato**), **descripción**, **disponibilidad**, **número de categoría**, **descripción de categoría**, **precio**, **receta**, **id** (**Primario**)

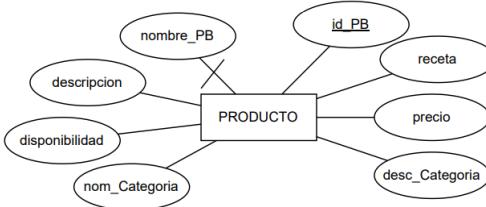


Figura 5: Entidad Producto

A su vez, el problema nos indica que dichos productos pueden ser platos o bebidas, por esta razón realizamos una especialización con exclusión donde usaremos un indicador llamado tipo para diferenciarlos uno de otro ya que no contienen atributos únicos.

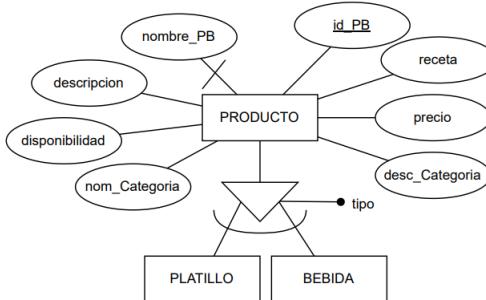


Figura 6: Especialización Producto

Para relacionar la entidad producto dentro de nuestro sistema hacemos uso de otra entidad definida como orden.

- **Orden** de la cual identificamos los atributos: **folio** (**Primario**), **fecha**, **total de orden** (**Calculado**)

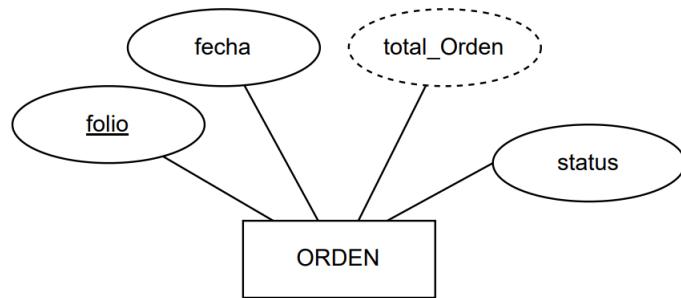


Figura 7: Entidad Orden

Dicha entidad se relaciona con producto mediante el verbo descriptivo “incluye” el cual tiene una cardinalidad M:M la cual se lee como 1 orden incluye Muchos productos y 1 producto puede ser incluido por Muchas órdenes.

Siguiendo con las relaciones de la entidad orden, tenemos que el mesero es el encargado de registrar dichas órdenes, por lo que usando el verbo descriptivo “registra” relacionamos ambas entidades con una cardinalidad 1:M, de tal manera que 1 mesero pueda registrar Muchas órdenes y 1 orden pueda ser registrada por 1 solo mesero.

Y para finalizar con las relaciones de orden tenemos que es necesario contar con un cliente el cual realice la orden, por lo que siguiendo la descripción del problema obtenemos la definición de la entidad cliente.

- **Cliente** identificando los siguientes atributos: **RFC (Primario)**, **nombre (Compuesto)**, **razón social**, **email**, **domicilio (Compuesto)**, **fecha de nacimiento (Calculada)**

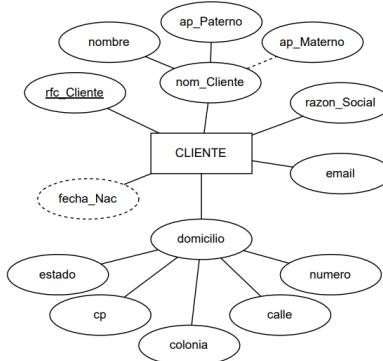


Figura 8: Entidad Cliente

Dicha relación ocurre con el verbo “realiza” y una cardinalidad de M:1 explicando que 1 orden puede ser realizada por 1 cliente y 1 cliente puede realizar Muchas órdenes.

3.2. Diagrama DER

Con las entidades y sus relaciones descritas anteriormente, realizamos el diagrama DER obteniendo como resultado

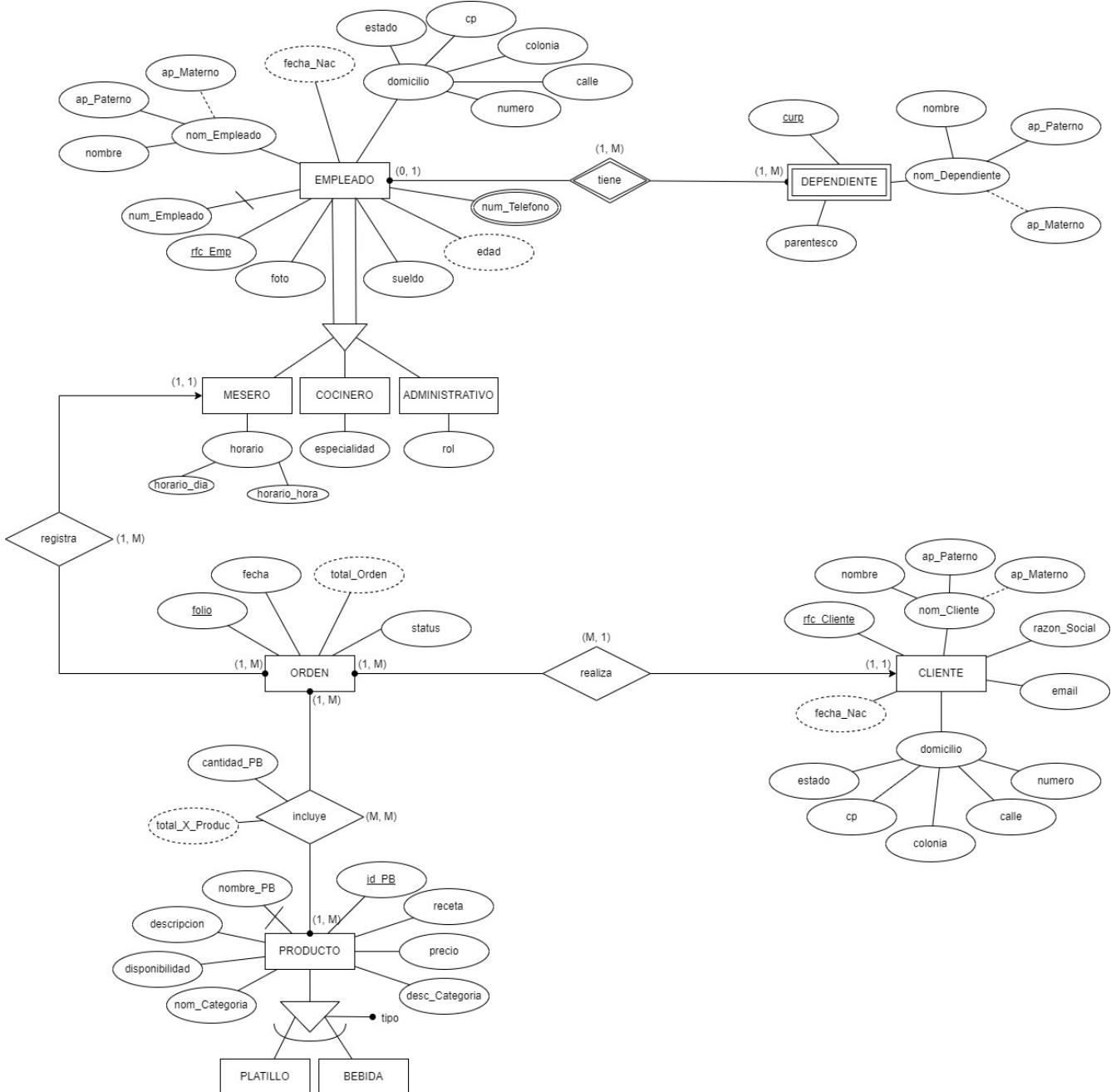


Figura 9: DER

3.3. Representación Intermedia y MR

Una vez obtenido el diagrama DER, hacemos uso de la 1º estrategia para separar las entidades de la especialización de empleado, de esta manera obtenemos.

EMPLEADO: {rfc_Emp varchar(13) (PK), num_Emppleado smallint (U), nombre varchar(50), ap_Paterno varchar(50), ap_Materno varchar(50) (N), fecha_Nac date (C), edad smallint (C), cp int, colonia varchar(40), calle

varchar(50), numero int, nombre_Estado varchar(30), sueldo money, foto text}



Figura 10: Representacion relacional de la entidad EMPLEADO

num_Teléfono: rfc_Emp varchar(13) (FK), telefono bigint (PK)



Figura 11: Representacion relacional del atributo multivaluado num_telefono

MESERO: {rfc_Emp varchar(13) (FK) (PK), horario_Dia varchar(5), horario_Hora varchar(3)}



Figura 12: Representacion relacional de la entidad MESERO

COCINERO: {rfc_Emp varchar(13) (FK) (PK), especialidad varchar(40)}



Figura 13: Representación relacional de la entidad COCINERO

ADMINISTRATIVO: {rfc varchar(13) (FK) (PK), rol varchar(40)}



Figura 14: Representacion relacional de la entidad ADMINISTRATIVO

Por otro lado, para la entidad debil "Dependiente" obtenemos la siguiente representación.

DEPENDIENTE: {curp varchar(18) (PK), rfc_Emp varchar(13) (FK), nombre varchar(50), ap_Paterno varchar(50), ap_Materno varchar(50) (N), parentesco varchar(30)}



Figura 15: Representacion relacional de la entidad DEPENDIENTE

Siguiendo nuestro diagrama DER, obtenemos las entidades relacionadas a "Cliente" de tal manera que:

CLIENTE: {rfc_Cliente varchar(13) (PK), nombre varchar(50), ap_Paterno varchar(50), ap_Materno varchar(50) (N), fecha_Nac date (C), razon_Social varchar(60), email varchar(100), cp int, colonia varchar(40), calle varchar(50), numero int, estado varchar(30)}



Figura 16: Representacion relacional de la entidad CLIENTE

ORDEN: {folio int (PK), fecha date, total_Orden int (C), rfc_Emp varchar(13) (FK), rfc_Cliente varchar(13) (FK)}



Figura 17: Representacion relacional de la entidad ORDEN

Mientras que para obtener la representación de la entidad producto se hace uso de la 4° estrategia para en una sola entidad representar la especialización bebida y platillo.

PRODUCTO: {id_PB int (PK), nombre_PB varchar(50) (U), precio money, disponibilidad bool, descripcion varchar(150), receta varchar(200), nom_Categoría varchar(50), desc_Categoría varchar(150), es_Platillo bool, es_Bebida bool}

public.PRODUCTO		
↳ <i>id_PB</i>	integer	« pk »
↳ <i>nombre_PB</i>	varchar	« uq »
↳ <i>precio</i>	money	« nn »
↳ <i>disponibilidad</i>	bool	« nn »
↳ <i>descripcion</i>	varchar(150)	« nn »
↳ <i>receta</i>	varchar(200)	« nn »
↳ <i>nom_Categoría</i>	varchar(50)	« nn »
↳ <i>es_Platillo</i>	bool	« nn »
↳ <i>es_bebida</i>	bool	« nn »
↳ <i>UNIQ_nom_PB</i>	constraint	« uq »
↳ <i>PK_PROD</i>	constraint	« pk »
↳ <i>CK_PLATILLO_BEBIDA</i>	constraint	« ck »

Figura 18: Representación relacional de la entidad PRODUCTO

Por último, debido a que tenemos una relación M:M, como tal no tenemos una tabla que la represente, ya que aunque no aparece en el modelo relacional Pgmodeler la crea, es decir sabemos que existe, pero no la vemos como tal.

INCLUYE: {[folio int (FK), id.PB int (FK)](PK), cantidad.PB smallint, total_X_Produc money (C)}

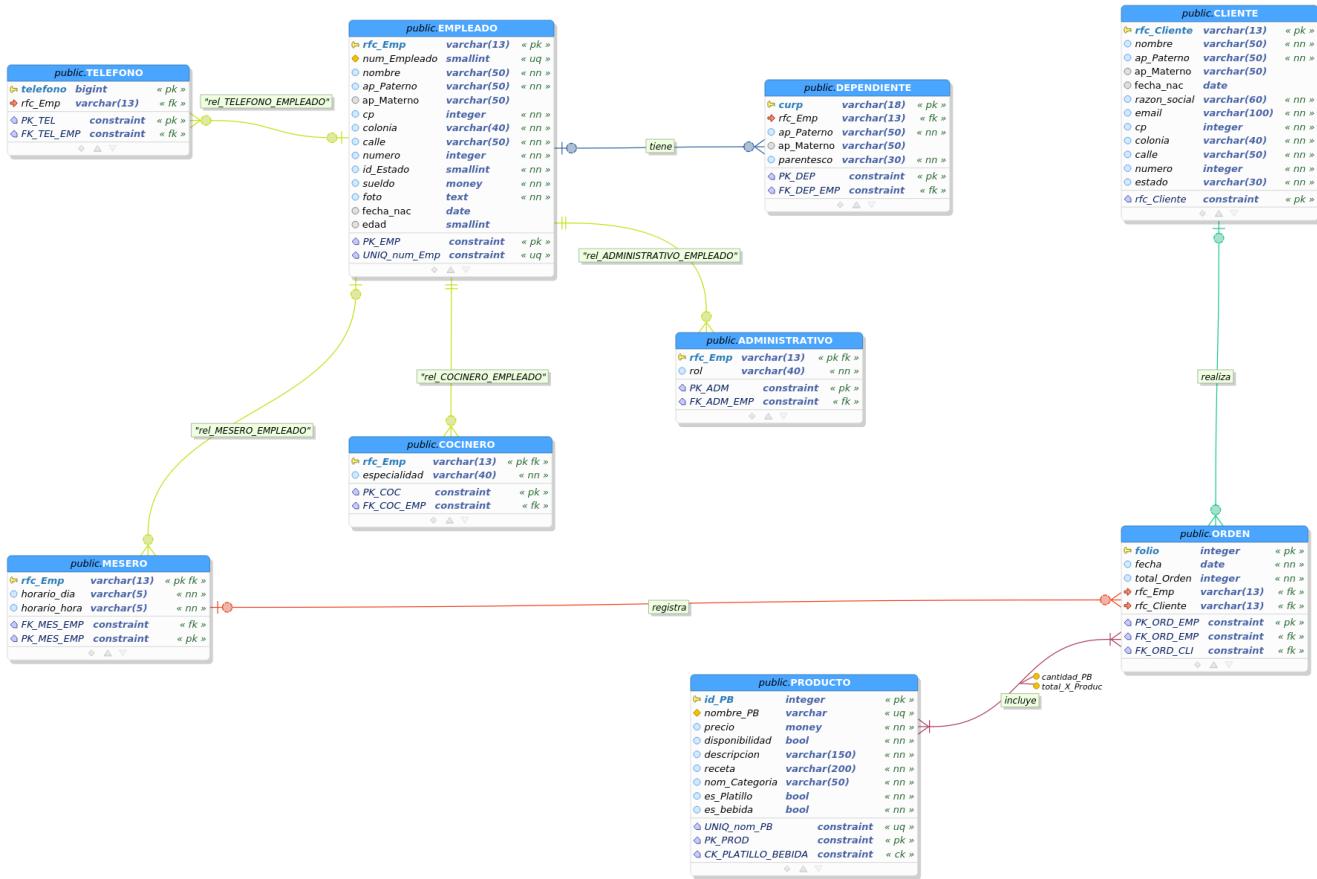


Figura 19: Modelo relacional

3.4. Normalización

Una vez bien definidas las tablas a utilizar dentro de nuestra base de datos, procedemos a realizar la normalización de cada una de ellas de tal manera que se describan las dependencias de sus atributos y se respondan a las preguntas correspondientes según sea la forma normal.

Para este proceso, normalizaremos hasta la 3º forma normal, obteniendo así los siguientes desarrollos.

Empleado

<u>rfc_Emp</u>	<u>num_Emp</u>	nombre	ap_Paterno	ap_Materno	fecha_Nacimiento	edad	cp	colonia	calle	numero	estado	sueldo	foto

The diagram shows the primary key rfc_Emp with blue arrows pointing to each column. A red vertical bar highlights the columns cp, colonia, calle, numero, estado, sueldo, and foto, indicating they are foreign keys pointing back to other tables.

1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

3FN

- Relaciones transitivas: Si

Por lo tanto NO cumple

Normalizando

PK							FK					
<u>rfc_Emp</u>	<u>num_Emp</u>	nombre	ap_Paterno	ap_Materno	fecha_Nacimiento	edad	<u>cp</u>	colonia	calle	numero	sueldo	foto
PK												
<u>cp</u>	colonia	estado										

Num_telefono

FK

<u>rfc_Emp</u>	<u>telefono</u>

The diagram shows the primary key rfc_Emp with a blue arrow pointing to the first column of the telefono row. A blue bracket connects the two rows, indicating a relationship between the primary key and the foreign key.

1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

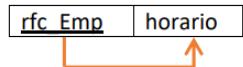
3FN

- Relaciones transitivas: No

Por lo tanto Cumple

Mesero

FK



1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

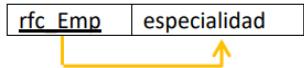
3FN

- Relaciones transitivas: No

Por lo tanto Cumple

Cocinero

FK



1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

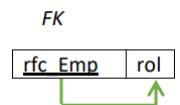
Por lo tanto Cumple

3FN

- Relaciones transitivas: No

Por lo tanto Cumple

Administrativo



1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

3FN

- Relaciones transitivas: No

Por lo tanto Cumple

Dependiente

FK



1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

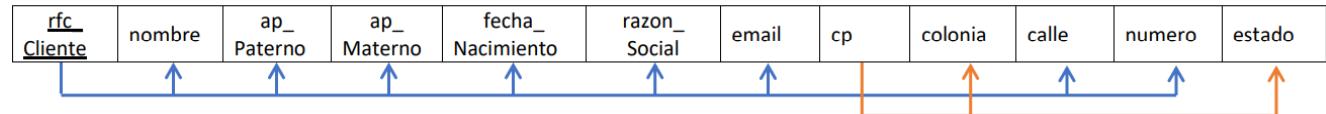
Por lo tanto Cumple

3FN

- Relaciones transitivas: No

Por lo tanto Cumple

Cliente



1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

3FN

- Relaciones transitivas: Si

Por lo tanto No cumple

Normalizando

PK										FK
<u>rfc</u> <u>Cliente</u>	nombre	ap_Paterno	ap_Materno	fecha_Nacimiento	razon_Social	email	cp	calle	numero	
PK										
cp	colonia	estado								

Orden

					FK	FK
folio	fecha	total_Orden	rfc_Emp	rfc_Cliente		
					↑	↑

1FN

- Atributos Multivaluados: No
- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

3FN

- Relaciones transitivas: No

Por lo tanto Cumple

Producto

<u>id_PB</u>	nombre	precio	disponibilidad	descripción	receta	nom_Categoría	desc_Categoría	es_Platillo	es_Bebiba

1FN

- Atributos Multivaluados: No

- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

3FN

- Relaciones transitivas: Si

Por lo tanto No cumple

Normalizando

<u>id_PB</u>	nombre	precio	disponibilidad	descripción	receta	nom_Categoría	es_Platillo	es_Bebiba
PK								
FK								
PK								
<u>nom_Categoría</u>		<u>desc_Categoría</u>						

Incluye

<u>folio</u>	<u>id_PB</u>	cantidad	precio

1FN

- Atributos Multivaluados: No

- Grupos de Repetición: No

Por lo tanto Cumple

2FN

- Dependencias funcionales parciales: No

Por lo tanto Cumple

3FN

- Relaciones transitivas: No

Por lo tanto Cumple

Después de aplicar la normalización, requerimos tener un nuevo modelo relacional que cumpla con la existencia de las nuevas tablas creadas:

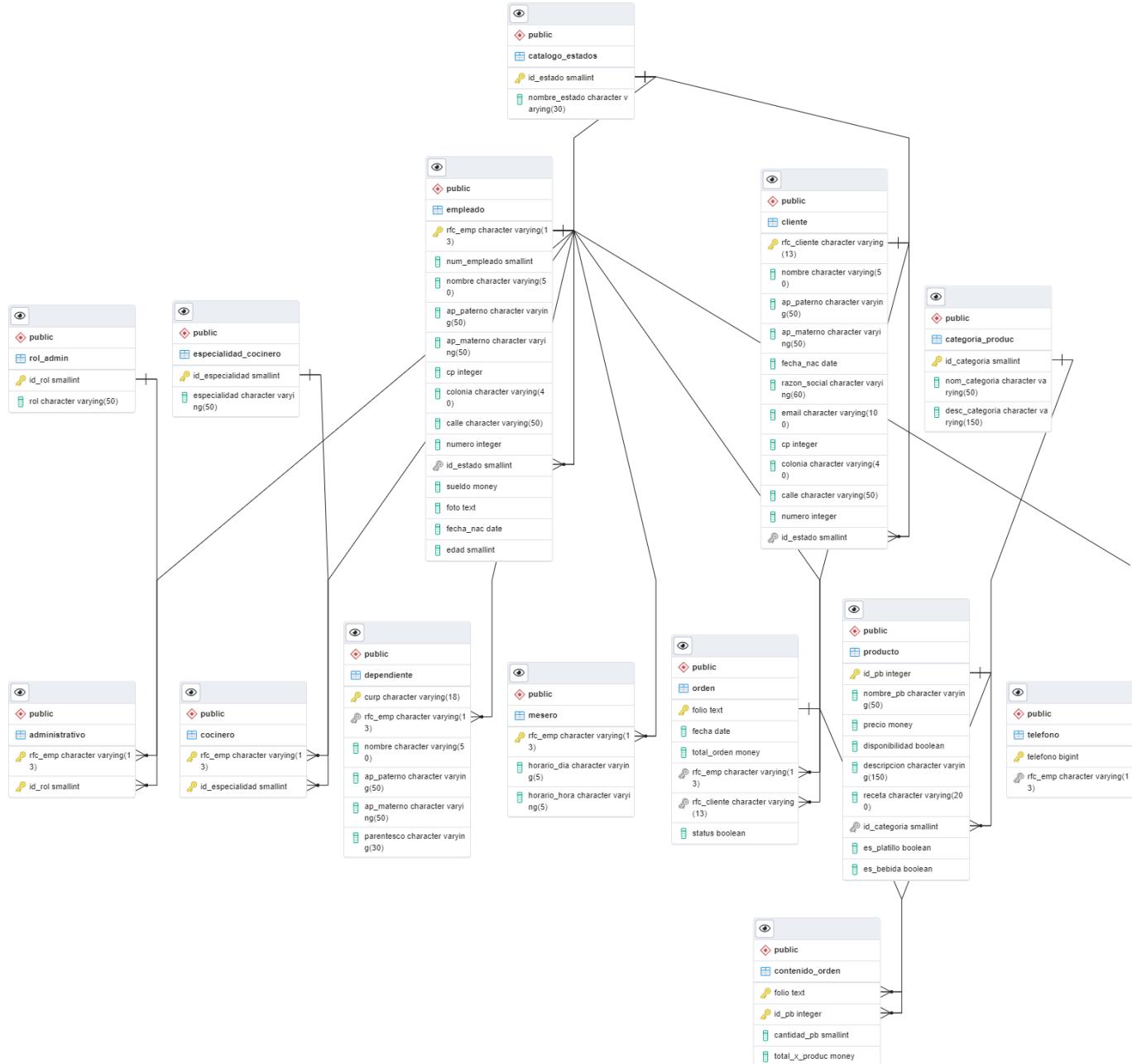


Figura 20: Modelo relacional final

4. Implementación

4.1. Creación de tablas

Una vez normalizadas las tablas, procedemos con su implementación usando DDL. Para la primera entidad "Empleado" observamos que se puede generar un catálogo de estados, por lo que cambiamos el atributo "nombre_Estado" por "id_Estado" para posteriormente hacer referencia a dicho catálogo.

Por otro lado, considerando nuestra representación intermedia, creamos los constraints para la llave primaria (RFC), un constraint de unique para el atributo candidato (num_Emp) y la referencia al catálogo descrito previamente como llaves foránea (id_Estados), agregando a este último la propiedad de "Delete" y "Update" en cascade para que de esta manera se mantenga actualizado con los movimientos realizados en la tabla de referencia. Dicho esto es importante crear primero la tabla CATALOGO_ESTADOS, ya que EMPLEADO esta haciendo referencia a esta y si no es creada antes nos marcará un error.

Cabe destacar que se decidió no seguir con la normalización en este caso la cual nos indica una posible generación de una tabla que incluya los atributos CP, colonia y Estado, ya que, como equipo consideramos que la probabilidad de registrar CP iguales es menor que la de registrar Estados iguales, por lo tanto es más óptimo solo tener un catálogo de Estados.

De esta manera tenemos el siguiente código:

```
CREATE TABLE EMPLEADO (
    rfc_Emp varchar(13), - LLAVE PRIMARIA
    num_Emp smallint , - CLAVE UNICA
    nombre varchar(50) NOT NULL,
    ap_Paterno varchar(50) NOT NULL,
    ap_Materno varchar(50),
    cp int NOT NULL,
    colonia varchar(40) NOT NULL,
    calle varchar(50) NOT NULL,
    numero int NOT NULL,
    id_Estado smallint, - ANTES nombre_Estado varchar(30)
    sueldo money NOT NULL,
    foto text NOT NULL,
    CONSTRAINT PK_EMP PRIMARY KEY(rfc_Emp),
    CONSTRAINT UNIQ_num_Emp UNIQUE(num_Emp),
    CONSTRAINT FK_EMP_CAT FOREIGN KEY(id_Estado) REFERENCES CATALOGO_ESTADOS(id_Estado)
    ON DELETE CASCADE on update cascade
);
```

Donde los campos calculados "fecha_Nac" y "edad" se agregan posteriormente de la inserción de datos mediante un archivo .csv usando la sentencia "ALTER"

```
ALTER TABLE EMPLEADO ADD fecha_Nac date, - TIPO CALCULADO
ALTER TABLE EMPLEADO ADD edad smallint, - TIPO CALCULADO
```

Por consiguiente, la creación de la tabla "Catalogo_Estados" queda descrita de la siguiente manera, donde se hace uso de un constraint de llave primaria para el atributo id_Estado.

```
CREATE TABLE CATALOGO_ESTADOS (
    id_Estado smallint,
    nombre_Estado varchar(30) NOT NULL, - ANTES estaba en la tabla EMPLEADO y CLIENTE
    CONSTRAINT PK_CAT_EST PRIMARY KEY(id_Estado)
);
```

Para la siguiente tabla, previamente definida como num_telefono, decidimos nombrarla como "telefono" para

facilitar su lectura y siguiendo nuestra representación intermedia, hacemos uso de constraints para definir la llave primaria(telefono) y la llave foránea(rfc_Emp) la cual hará referencia a la tabla "Empleado" aplicando "Delete Cascade" y "Update Cascade" para mantener la sincronización entre ambas tablas.

```
CREATE TABLE TELEFONO (
telefono bigint,
rfc_Emp varchar(13),
CONSTRAINT PK_TEL PRIMARY KEY(telefono),
CONSTRAINT FK_TEL_EMP FOREIGN KEY(rfc_Emp) REFERENCES EMPLEADO(rfc_Emp) ON DELETE CASCADE on update cascade
);
```

Siguiendo el mismo proceso creamos la tabla "Mesero" y sus respectivos constraints, donde la llave primaria y foránea hacen referencia ambas al atributo rfc_Emp, con la diferencia de que el constraint de llave foránea agrega las sentencias "Delete/Update Cascade".

```
CREATE TABLE MESERO (
rfc_Emp varchar(13),
horario_Dia varchar (5) NOT NULL,
horario_Hora varchar (5) NOT NULL,
CONSTRAINT PK_MES PRIMARY KEY(rfc_Emp),
CONSTRAINT FK_MES_EMP FOREIGN KEY(rfc_Emp) REFERENCES EMPLEADO(rfc_Emp) ON DELETE CASCADE on update cascade
);
```

En el caso de la tabla "Cocinero" decidimos crear un catálogo para almacenar la especialidad, por lo que el atributo "especialidad" de nuestra representación intermedia se convertirá en "id_Especialidad", agregando un constraint de llave foránea para enlazar la tabla con el catálogo.

Y de igual forma que la tabla anterior "Mesero" se define como llave primaria y llave foránea al rfc_Emp con sus correspondientes constraints.

```
CREATE TABLE COCINERO (
rfc_Emp varchar(13),
id_Especialidad smallint, -ANTES era especialidad varchar(40) Se convertirá en catálogo
CONSTRAINT PK_COX PRIMARY KEY(rfc_Emp),
CONSTRAINT FK_COX_EMP FOREIGN KEY(rfc_Emp) REFERENCES EMPLEADO(rfc_Emp) ON DELETE CASCADE on update cascade,
CONSTRAINT FK_COX_ESP FOREIGN KEY(id_Especialidad) REFERENCES
ESPECIALIDAD_COX(id_Especialidad) ON DELETE CASCADE on update cascade
);
```

Donde el catálogo descrito previamente tiene el nombre de "Especialidad_Cocinero".

```
CREATE TABLE ESPECIALIDAD_COX (
id_Especialidad smallint,
Especialidad varchar(50), -ANTES estaba en la tabla COCINERO
CONSTRAINT PK_ESP_COX PRIMARY KEY(id_Especialidad)
);
```

Siguiendo la misma lógica, para la tabla "Administrativo" separamos el atributo "rol" en un catálogo mediante la llave foránea "id_rol" y realizamos los constraints correspondientes para el atributo "rfc_Emp".

```
CREATE TABLE ADMINISTRATIVO (
rfc_Emp varchar(13),
```

```

id_Rol smallint, -ANTES era rol varchar(40) Se convertirá en catálogo
CONSTRAINT PK ADM PRIMARY KEY(rfc_Emp),
CONSTRAINT FK ADM EMP FOREIGN KEY(rfc_Emp) REFERENCES EMPLEADO(rfc_Emp) ON DELETE
CASCADE on update cascade,
CONSTRAINT FK ADM ROL FOREIGN KEY(id_Rol) REFERENCES ROL ADMIN(id_Rol) ON DELETE CAS-
CADE on update cascade
);

```

Y por su parte, creamos el catálogo con el nombre de "Rol_Admin" y su respectiva llave primaria(id_Rol) mediante un constraint.

```

CREATE TABLE ROL_ADMIN (
id_Rol smallint,
rol varchar(50), -ANTES estaba en la tabla ADMINISTRATIVO
CONSTRAINT PK ROL ADMIN PRIMARY KEY(id_Rol)
);

```

Prosiguiendo con la entidad "Dependiente" usamos la definición descrita en la representación intermedia agregando a esta los constraints para la llave primaria(curp) y la llave foránea(rfc_Emp) la cual nos permite enlazar la tabla "Empleado" y esta.

```

CREATE TABLE DEPENDIENTE (
curp varchar(18), -LLAVE PRIMARIA
rfc_Emp varchar(13),
nombre varchar(50) NOT NULL,
ap_Paterno varchar(50) NOT NULL,
ap_Materno varchar(50),
parentesco varchar(30) NOT NULL,
CONSTRAINT FK DEP EMP FOREIGN KEY(rfc_Emp) REFERENCES EMPLEADO(rfc_Emp) ON DELETE
CASCADE on update cascade,
CONSTRAINT PK DEP PRIMARY KEY(curp)
);

```

De la misma forma que "Empleado", para la tabla "Cliente" hacemos uso del catálogo "Catalogo_Estados" por lo que realizamos el cambio de atributo "Estado" por "id_Estado" mismo que agregamos a un constraint de llave foránea para referenciarlo en el catálogo.

En este caso, el campo calculado "fecha_Nac" se incluye en la creación de la tabla y no en un alter como lo es en caso de la tabla "Empleado", ya que para esta tabla la inserción de datos no se realiza mediante un archivo .csv.

```

CREATE TABLE CLIENTE (
rfc_Cliente varchar(13),
nombre varchar(50) NOT NULL,
ap_Paterno varchar(50) NOT NULL,
ap_Materno varchar(50),
fecha_Nac date (C), - TIPO CALCULADO
razon_Social varchar(60) NOT NULL,
email varchar(100) NOT NULL,
cp int NOT NULL,
colonia varchar(40) NOT NULL,
calle varchar(50) NOT NULL,
numero int NOT NULL,
id_Estado smallint, - Ya es un catálogo
CONSTRAINT PK CLIENTE PRIMARY KEY(rfc_Cliente),

```

```

CONSTRAINT FK_CLI_CAT FOREIGN KEY(id_Estado) REFERENCES CATALOGO_ESTADOS(id_Estado)
ON DELETE CASCADE on update cascade
);

```

Siguiendo nuestra representación intermedia obtenemos la tabla "Orden" en la cual agregamos los constraints necesarios para indicar la llave primaria(folio) y las llaves foráneas(rfc_Emp y rfc_Cliente) con sus correspondientes referencias a las tablas "Empleado" y "Cliente", así como las sentencias "Delete/Update Cascade".

Para el atributo fecha indicamos que el valor de defecto será el actual al momento de realizar la inserción para facilitar su registro. Por otro lado, agregamos un atributo "status" de tipo booleano el cual será utilizado por la interfaz para realizar el cierre de cuenta.

```

CREATE TABLE ORDEN (
folio text, -ORD-001 Donde 001 es un numero secuencial
fecha date default now(),
total_Orden money, -TIPO CALCULADO, Suma de los totales por producto, es decir suma n total_X_Produc
rfc_Emp varchar(13),
rfc_Cliente varchar(13),
status BOOLEAN, -Usado por la interfaz para cerrar cuentas
CONSTRAINT PK_ORD PRIMARY KEY(folio),
CONSTRAINT FK_ORD_EMP FOREIGN KEY(rfc_Emp) REFERENCES EMPLEADO(rfc_Emp) ON DELETE
CASCADE on update cascade,
CONSTRAINT FK_ORD_CLI FOREIGN KEY(rfc_Cliente) REFERENCES CLIENTE(rfc_Cliente) ON DELETE
CASCADE on update cascade
);

```

Para la creación de la tabla "Producto" agregamos constraints con los cuales podemos definir la llave primaria(id_PB), la llave candidata o única(nombre_PB), la llave foránea(id_Categoría) con referencia a un catálogo bajo el nombre de "Categoria_Produc" y la restricción Check para los atributos "es_Platillo" y "es_Bebida" la cual nos indica que estos deben de ser distintos entre sí, por lo que al ser de tipo booleano nos denota que un producto puede ser platillo o bebida pero no ambos.

```

CREATE TABLE PRODUCTO (
id_PB int,
nombre_PB varchar(50) NOT NULL, -Tipo unico
precio money NOT NULL, -Precio es "el precio unitario del producto"
disponibilidad bool NOT NULL,
descripcion varchar (150) NOT NULL,
receta varchar(200) NOT NULL,
id_Categoría smallint, -ANTES era nom_Categoría varchar(50)
es_Platillo bool NOT NULL,
es_Bebida bool NOT NULL,
CONSTRAINT PK_PROD PRIMARY KEY(id_PB),
CONSTRAINT UNIQ_nom_PB UNIQUE(nombre_PB),
CONSTRAINT FK_PRO_CAT FOREIGN KEY(id_Categoría) REFERENCES CATEGORIA_PRODUC(id_Categoría)
ON DELETE CASCADE on update cascade,
CONSTRAINT CK_PLATILLO_BEBIDA CHECK(es_Platillo != es_Bebida)
);

```

Como previamente indicamos, creamos una referencia a un catálogo "Categoria_Produc" el cual contiene el nombre y descripción de la categoría a la cual pertenece cada producto, donde la llave primaria(id_Categoría) está indicada con un constraint de primary key.

```
CREATE TABLE CATEGORIA_PRODUC (
```

```

id_Categoría smallint,
nom_Categoría varchar(50) NOT NULL, -ANTES estaba en la tabla PRODUCTO
desc_Categoría varchar(150) NOT NULL, -ANTES estaba en la tabla PRODUCTO
CONSTRAINT PK_CAT_PRODUC PRIMARY KEY(id_Categoría)
);

```

Por último, tenemos la tabla para la relación M:M "incluye" la cual renombramos como "Contenido_Orden" para una mejor lectura. De igual manera que en las anteriores, definimos la llave primaria compuesta(folio,id_PB) y las llaves foráneas(folio y id_PB) usando constraints.

```

CREATE TABLE CONTENIDO_ORDEN (
folio text, -ORD-001 Donde 001 es un numero secuencial
id_PB int,
cantidad_PB smallint NOT NULL, -Cantidad por producto
total_X_Produc money, -TIPO CALCULADO (precio del producto unitario (precio)* cantidad que pide (cantidad_PB) )
CONSTRAINT FK_CONTENIDO_ORD FOREIGN KEY(folio) REFERENCES ORDEN(folio) ON DELETE CASCADE on update cascade,
CONSTRAINT FK_CONT_PRODUC FOREIGN KEY(id_PB) REFERENCES PRODUCTO(id_PB) ON DELETE CASCADE on update cascade,
CONSTRAINT PK_CONT_ORD PRIMARY KEY(folio, id_PB)
);

```

4.2. Campos Calculados (Triggers y Funciones)

4.2.1. Secuencias

Para facilitar la inserción de datos a la tabla empleado, se decidió crear una secuencia que lleve la cuenta del número de empleado que se desea registrar. Se comienza desde el 11 porque ya insertamos 10 registros justo después de crear la tabla.

Para esto, se hizo uso del siguiente código en SQL.

```

CREATE SEQUENCE cuenta_num_emp
START WITH 11
INCREMENT BY 1;

```

Figura 21: Secuencia cuenta_num.emp

De igual forma, generamos una secuencia para el número de folio asignado a las órdenes creadas, cabe destacar que este número se concatena con la cadena "ORD-" para unificar el formato de las órdenes.

```

CREATE SEQUENCE genera_Folio
START WITH 001
INCREMENT BY 1;

```

Figura 22: Secuencia genera_folio

4.2.2. Fecha de nacimiento y Edad

Para obtener la fecha de nacimiento y por consiguiente la edad de un empleado, se hace uso de su RFC el cual está formado por los primeros 5 caracteres que hacen referencia al nombre completo seguido de 6 caracteres definiendo la fecha de nacimiento de tipo Año/Mes/Dia.

Una vez establecido la lógica anterior, procedemos a crear una función llamada "calc_Fecha_Edad" la cual actualizara los campos de fecha de nacimiento y edad mediante un Update. De tal manera que para la fecha de nacimiento se obtengan los 6 caracteres de Año/Mes/Dia de la cadena rfc_Emp usando el comando substring(rfc_Emp,5,6), donde 5 es el carácter inicial y 6 la longitud; y se conviertan de tipo cadena a tipo fecha usando CAST().

Posteriormente, usando EXTRACT(YEAR FROM CURRENT_DATE) se obtiene el año de la fecha actual y se le resta el año de la fecha almacenada en fech_Nac usando igualmente un EXTRACT()

```
CREATE FUNCTION calc_Fecha_Edad() RETURNS trigger AS $fecha_edad$  
BEGIN  
    UPDATE EMPLEADO SET fecha_Nac = CAST(substring (rfc_Emp, 5,6) AS date) WHERE fecha_Nac IS NULL;  
    UPDATE EMPLEADO SET edad = EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM fecha_Nac) WHERE edad IS NULL;  
    RETURN new;  
END;  
$fecha_edad$ LANGUAGE plpgsql;
```

Figura 23: Función calc_fecha_edad

A fin de que la función previamente descrita se ejecute de forma automática cada vez que se inserte un nuevo cliente, procedemos a realizar un trigger cuyo nombre es "insert_fecha_edad" el cual nos indica que después de cada inserción a la tabla de empleados (AFTER INSERT ON EMPLEADO) se ejecute la función para cada registro (FOR EACH ROW EXECUTE PROCEDURE).

```
CREATE TRIGGER insert_fecha_edad AFTER INSERT ON EMPLEADO  
FOR EACH ROW EXECUTE PROCEDURE calc_Fecha_Edad();
```

Figura 24: Trigger insertar_fecha_edad

Debido a que para la entidad "Cliente" no se cuenta con el atributo "Edad", la función descrita previamente no se puede reutilizar en su totalidad.

Por lo que solo se ocupa la misma lógica para obtener la fecha de nacimiento a partir de un rfc, obteniendo así una nueva función "calc_Fecha_Cliente".

```
CREATE FUNCTION calc_Fecha_cliente() RETURNS trigger AS $fecha$  
BEGIN  
    UPDATE CLIENTE SET fecha_Nac = CAST(substring (rfc_Cliente, 5,6) AS date) WHERE fecha_Nac IS NULL;  
    RETURN new;  
END;  
$fecha$ LANGUAGE plpgsql;
```

Figura 25: Función calc_fecha_cliente

De igual forma, se crea un trigger encargado de ejecutar la función "calc_Fecha_Cliente" después de que se realice un insert en la tabla "Cliente", dicho trigger recibe el nombre de "insert_fecha_cliente".

```
CREATE TRIGGER insert_fecha_cliente AFTER INSERT ON CLIENTE  
FOR EACH ROW EXECUTE PROCEDURE calc_Fecha_cliente();
```

Figura 26: Trigger insertar_fecha_cliente

4.2.3. Generar Ordenes

Para generar y asignar un mesero a una orden se hace uso de la función "cliente_genera_orden" la cual dada una fecha (actual) y un rfc de un cliente (rfc_Ingresado) procede primero a buscar si existe el registro del rfc en la tabla

de cliente, en caso de no existir se manda un mensaje con la leyenda "No se encontró el rfc"; en caso de comprobar su existencia se le asigna un mesero aleatorio, para realizar esto usamos el comando "random()" para seleccionar al azar un rfc de la tabla Mesero.

Posteriormente, usando la secuencia "genera_Folio" lo concatenamos con la cadena "ORD-" y de esta manera obtenemos el folio a asignar para la orden.

Para finalizar, se insertan los valores obtenidos a la tabla ORDEN siguiendo la estructura de VALUES(folio, fecha, 0,rfc_Mesero,rfc_Cliente,True), donde el 0 nos indica el total de la orden y "True" el estado de la orden donde True es abierta y False es cerrada.

```
CREATE FUNCTION cliente_genera_orden(fecha text,rfc_Ingresado text) RETURNS void AS $generaOrden$  
DECLARE  
mesero_Aleatorio varchar(13);  
folio text;  
BEGIN  
IF EXISTS (SELECT rfc_Cliente FROM CLIENTE WHERE rfc_Cliente = rfc_Ingresado ) THEN  
mesero_Aleatorio := (SELECT rfc_Emp FROM MESERO order by random() limit 1);  
folio:= (SELECT CONCAT('ORD','-',NEXTVAL('genera_Folio')));  
INSERT INTO ORDEN VALUES(folio,cast(fecha as date),0,mesero_Aleatorio,rfc_Ingresado,True);  
RAISE NOTICE 'Se genero la orden: %', folio;  
ELSE  
RAISE NOTICE 'No se encontro el rfc: %', rfc_Ingresado;  
END IF;  
END;  
$generaOrden$ LANGUAGE plpgsql;
```

Figura 27: Función cliente_genera_orden

4.2.4. Agregar Productos

En el caso de querer agregar productos a las ordenes registradas, hacemos uso de una función "agrega_productos" la cual recibe como parámetros el folio de la orden, el id del producto a agregar y la cantidad de dicho producto.

Primero se comprueba la existencia del folio en la tabla ORDEN usando un SELECT y la sentencia IF EXISTS, en caso de que no exista dicho folio se manda un mensaje indicando el error. Se prosigue por verificar si existe una referencia del id del producto en la tabla PRODUCTO y de igual manera, haciendo uso de SELECT y IF EXISTS, si no se cumple se manda un mensaje de error.

Una vez comprobada la existencia del folio y el id del producto, se procede a comprobar si dicho producto se encuentra disponible, para esto se realiza un SELECT en la tabla PRODUCTO, donde la disponibilidad debe de tener un valor True.

```
CREATE FUNCTION agrega_productos(folio_ingresado text, id_producto integer, cantidad integer) RETURNS void AS $agrega$  
DECLARE  
v_precio_producto money;  
v_precio_X_cantidad money;  
BEGIN  
IF EXISTS(SELECT folio FROM ORDEN WHERE folio = folio_ingresado) THEN  
RAISE NOTICE 'Sí existe el folio: %', folio_ingresado;  
IF EXISTS(SELECT id_PB FROM PRODUCTO WHERE id_PB = id_producto) THEN  
RAISE NOTICE 'Sí existe el id de producto: %', id_producto;  
IF EXISTS(SELECT disponibilidad FROM PRODUCTO WHERE id_PB = id_producto AND disponibilidad = TRUE) THEN  
RAISE NOTICE 'Sí hay disponibilidad del producto: %', id_producto;
```

Figura 28: Función agregar_productos - 1

En caso de que las tres condiciones anteriores se hayan cumplido, es decir, exista el folio, el producto y este esté

disponible, se comprueba si dentro de la orden el producto que se quiere registrar ya se encuentra registrado o no.

En caso de estar ya registrado primero se actualizara la cantidad sumándole la cantidad ingresada con un UPDATE en el atributo "cantidad_pb", como segundo paso se obtiene y guarda el precio del producto en una variable "v_precio_producto" la cual se ocupara para multiplicarse por la cantidad ingresada (La que recibe la función como parámetro). Este dato se sumara al total del producto mediante un UPDATE en el atributo total_X_produc y de igual manera, con un UPDATE a "total_orden" se suma al total de la orden.

```
IF EXISTS(SELECT id_pb FROM CONTENIDO_ORDEN WHERE id_pb = id_producto AND folio = folio_ingresado)THEN
    UPDATE CONTENIDO_ORDEN SET cantidad_pb = cantidad_pb + cantidad WHERE id_pb = id_producto AND folio = folio_ingresado;
    v_precio_producto:=(SELECT precio FROM PRODUCTO WHERE id_PB = id_producto);
    v_precio_X_cantidad:= v_precio_producto * cantidad;
    UPDATE CONTENIDO_ORDEN SET total_X_produc = total_X_produc + v_precio_X_cantidad WHERE id_pb = id_producto AND folio = folio_ingresado;
    UPDATE ORDEN SET total_orden = v_precio_X_cantidad + total_orden WHERE folio = folio_ingresado;
```

Figura 29: Función agregar_productos - 2

En caso de que el producto no se encuentre registrado, se aplica una lógica similar, primero se obtiene el precio del producto y se multiplica por la cantidad ingresada para con ese valor obtenido actualizar el total de la orden y a diferencia de caso anterior, se realiza un INSERT para agregar el id del producto, la cantidad y el total del producto a la orden.

Por último, se muestra un mensaje de confirmación 'El total a pagar por el producto es'.

```
ELSE
    v_precio_producto:=(SELECT precio FROM PRODUCTO WHERE id_PB = id_producto);
    v_precio_X_cantidad:= v_precio_producto * cantidad;
    UPDATE ORDEN SET total_orden = v_precio_X_cantidad + total_orden WHERE folio = folio_ingresado;
    INSERT INTO CONTENIDO_ORDEN VALUES(folio_ingresado,id_producto,cantidad,v_precio_X_cantidad);
    RAISE NOTICE 'El total a pagar por el producto es: %', v_precio_X_cantidad;
END IF;
ELSE
    RAISE NOTICE 'No hay disponibilidad del producto: %', id_producto;
END IF;
ELSE
    RAISE NOTICE 'No existe el id de producto: %', id_producto;
END IF;
ELSE
    RAISE NOTICE 'No existe el folio: %', folio_ingresado;
END IF;
END;
$agrega$ LANGUAGE plpgsql;
```

Figura 30: Función agregar_productos - 3

4.2.5. Cantidad ordenes por mesero

Para conocer la cantidad de ordenes que ha concretado un mesero a partir de su número de identificación usamos una función con nombre "cantidad_ordenes_mesero" para obtener el rfc asociado a un mesero.

Dicha función realiza primero la verificación de existencia del número de empleado dentro de la tabla Empleado, en caso de que si exista, se toma el valor del rfc asociado y se procede a verificar si dicho rfc se encuentra dentro de la tabla Mesero.

En caso de que ambas condiciones se cumplan, se regresa el rfc del mesero para posteriormente dentro de la interfaz realizar la operación COUNT(folio) de las ordenes asociadas al rfc del Mesero que obtuvimos.

En caso contrario, se manda a pantalla los mensajes de error correspondientes.

```

CREATE FUNCTION cantidad_ordenes_mesero(numero_emp integer) RETURNS varchar AS $body$
DECLARE
busca_rfc VARCHAR(13);
BEGIN
IF EXISTS(SELECT num_empleado FROM EMPLEADO WHERE num_empleado = numero_emp) THEN
RAISE NOTICE 'Sí existe el numero de empleado: %',numero_emp;
busca_rfc :=(SELECT rfc_emp FROM EMPLEADO WHERE num_empleado = numero_emp);
IF EXISTS(SELECT rfc_emp FROM MESERO WHERE rfc_emp = busca_rfc) THEN
RAISE NOTICE 'El empleado ingresado sí es mesero';
RETURN busca_rfc;
ELSE
RAISE NOTICE 'El empleado ingresado NO es mesero';
END IF;
ELSE
RAISE NOTICE 'No existe el numero de empleado: %',numero_emp;
END IF;
END;
$body$ LANGUAGE plpgsql;

```

Figura 31: Función cantidad_ordenes_mesero

La consulta que nosotros utilizamos para probarla en el manejador fue:

```

SELECT COUNT(folio) AS cantidad_de_ordenes_generadas, SUM(total_orden) FROM ORDEN
WHERE rfc_emp = cantidad_ordenes_mesero(7) AND fecha = CURRENT_DATE; --EL 07 ES VARIABLE CONFORME AL num_empleado

```

Figura 32: Función cantidad_ordenes_mesero

donde lo subrayado es lo que se cambia para buscar a diferentes meseros, esto evidentemente se tiene que colocar, pero no en esta consulta sino en la interfaz.

4.2.6. Bebida y Platillo más vendido

Si se desea obtener el platillo más vendido se hace uso de la función ”platillo_mas_vendido”

La lógica detrás de la función se basa en que cada id del producto esté compuesto por: num_Categoría 'X 00 Y': Donde X corresponde a la categoría a la que pertenece el producto que puede ser alguna de las siguientes:

Cuadro 2: Código asignado por categoría

Primer digito del id_PB	Categoría
1	Entradas
2	Al pastor
3	Alambres
4	A la parrilla
5	Sopas
6	Bebidas
7	Bar
8	Postres
9	Extras

y 'Y' es el producto específico que pertenece a esa categoría.

Por esta razón, si se desea delimitar las categorías, se puede asumir que cada 1000 es el inicio y fin de una de ellas.

Conociendo lo anterior, realizamos una primera iteración donde un contador j va de 1 al número máximo de categorías (en este caso 9), se definen límites inferiores y superiores para obtener los id de los productos dentro de esta categoría y por cada iteración a dichos límites se les suma 1000.

Se hace uso de una variable "iteraciones" la cual guarda la cantidad de productos que se encuentran dentro de los límites mencionados. Para realizar esto se hace uso de COUNT(id_pb) dentro de un SELECT con las condiciones "id_pb mayor que limite_inf e id_pb menor que limite_superior", así como "es_platillo = True" para obtener solamente los productos que se encuentren dentro de los límites y que sean platos.

```

CREATE FUNCTION platillo_mas_vendido() RETURNS integer AS $body$
DECLARE
i integer:= 1;
j integer:= 1;
iteraciones integer;
id_iteracion integer :=1000;
id_mayor integer:=0;
cantidad integer:=0; --cantidad del producto más vendido
limite_inf integer:=1000;
limite_sup integer:=2000;
BEGIN
  FOR j IN 1..9 LOOP
    IF j > 1 THEN
      limite_inf:=limite_inf + 1000;
      limite_sup:=limite_sup + 1000;
    END IF;
    iteraciones:=(SELECT COUNT(id_pb) FROM PRODUCTO WHERE id_pb > limite_inf AND id_pb < limite_sup AND es_platillo = true);
    id_iteracion := limite_inf;
    -- ...
  END LOOP;
END;
$body$;
  
```

Figura 33: Función platillo_mas_vendido - 1

Una vez obtenido la cantidad de productos dentro de la categoría, se realiza un segundo loop de 1 a la cantidad de productos (iteraciones), donde id_iteraciones fungirá como un apuntador que nos ayude a recorrer los id de los productos, por lo que se inicializa con el límite inferior y por cada iteración se le sumará 1.

Dentro del segundo loop se comprobará la existencia del id del producto dentro de la tabla CONTENID_ORDEN, en caso de encontrar dicho producto se extraerá la cantidad total de veces que se ha pedido usando SUM(cantidad_pb).

Posteriormente se comprueba si dicho valor obtenido es mayor a una variable auxiliar "cantidad", en caso de serlo, se actualiza el valor de "cantidad" al encontrado y se guarda una referencia del id del producto en otra variable auxiliar "id_mayor".

Si no es mayor a "cantidad" se prosigue con los loops hasta terminar de recorrer todas las categorías. Finalmente, la función regresa el id del producto almacenado en "id_mayor" el cual hace referencia al plato más vendido.

```

FOR i IN 1..iteraciones LOOP
    id_iteracion:= id_iteracion + 1;
    IF EXISTS (SELECT id_pb FROM CONTENIDO_ORDEN WHERE id_pb = id_iteracion) THEN
        IF ((SELECT SUM(cantidad_pb) FROM CONTENIDO_ORDEN WHERE id_pb = id_iteracion) > cantidad) THEN
            cantidad:=(SELECT SUM(cantidad_pb) FROM CONTENIDO_ORDEN WHERE id_pb = id_iteracion);
            id_mayor := id_iteracion;
            RAISE NOTICE ' cantidad = %', cantidad;
            RAISE NOTICE ' id_mayor = %', id_mayor;
        END IF;
    END IF;
    END LOOP;
    RETURN id_mayor;
END;
$body$ LANGUAGE plpgsql;

```

Figura 34: Función platillo_mas_vendido - 2

Debido a que la lógica respecto a encontrar el platillo más vendido y encontrar la bebida más vendida es la misma, con excepción a una de las condiciones a verificar, donde para el platillo se busca que el atributo "es_platillo" sea True, para obtener la bebida se busca que "es_bebida" sea True.

```

CREATE FUNCTION bebida_mas_vendido() RETURNS integer AS $body$
DECLARE
i integer:= 1;
j integer:= 1;
iteraciones integer;
id_iteracion integer :=1000;
id_mayor integer:=0;
cantidad integer:=0; --cantidad del producto más vendido, solo se cambia si el id existe y si la suma de la cantidad es mayor a la cantidad previa
limite_inf integer:=6006;
limite_sup integer:=6009;
BEGIN
    FOR j IN 1..2 LOOP
        IF j > 1 THEN
            limite_inf:=limite_inf + 995;
            limite_sup:=limite_sup + 994;
        END IF;
        iteraciones:=(SELECT COUNT(id_pb) FROM PRODUCTO WHERE id_pb > limite_inf AND id_pb < limite_sup AND es_bebida = true);
        id_iteracion := limite_inf;
        FOR i IN 1..iteraciones LOOP
            id_iteracion:= id_iteracion + 1;
            RAISE NOTICE 'Entra %',id_iteracion;
            IF EXISTS (SELECT id_pb FROM CONTENIDO_ORDEN WHERE id_pb = id_iteracion) THEN
                RAISE NOTICE 'Entra %',i;
                IF ((SELECT SUM(cantidad_pb) FROM CONTENIDO_ORDEN WHERE id_pb = id_iteracion) > cantidad) THEN
                    cantidad:=(SELECT SUM(cantidad_pb) FROM CONTENIDO_ORDEN WHERE id_pb = id_iteracion);
                    id_mayor := id_iteracion;
                    RAISE NOTICE ' cantidad = %', cantidad;
                END IF;
            END IF;
        END LOOP;
    END LOOP;
    RETURN id_mayor;
END;
$body$ LANGUAGE plpgsql;

```

Herramienta Recortes

```

        RAISE NOTICE ' id_mayor = %', id_mayor;
    END IF;
END IF;
END LOOP;
END LOOP;
RETURN id_mayor;
END;
$body$ LANGUAGE plpgsql;

```

Figura 35: Función bebida_mas_vendido

4.3. Inserción de datos en el manejador

4.3.1. Archivos .csv

Para realizar los inserts, debido a que es una cantidad grande de datos para cada tabla, por practicidad se decidió hacer uso de archivos .csv para cargar dichos datos.

Por lo que haciendo uso del comando `COPY nombreTabla FROM 'ruta' USING DELIMITERS ',';` se realizó la carga de los archivos, la coma es el carácter que nos indica que se está pasando a otra columna del registro:

Empleados.csv

The screenshot shows the pgAdmin interface. On the left, there is a schema browser window titled 'public' containing the 'EMPLEADO' table. The table has 14 columns: rfc_Emp, num_Emp, nombre, ap_Paterno, ap_Materno, cp, colonia, calle, numero, id_Estado, sueldo, foto, fecha_nac, and edad. Primary key constraints ('PK_EMP') and unique key constraints ('UNIQ_num_Emp') are defined. On the right, a preview window displays the first 10 rows of the 'empleados.csv' file. The data includes columns such as ID, Nombre, Apellido Paterno, Apellido Materno, Colonia, Calle, Numero, Estado, Sueldo, Foto, Fecha_Nac, and Edad. Each row also contains a unique identifier at the beginning.

Figura 36: EMPLEADOS.CSV

Como pudimos apreciar en la Figura 36, no agregamos la fecha_Nac ni la edad, ya que al insertar los datos en nuestro manejador no se hace la forma de insertar NULL, por lo que se realiza después un ALTER hacia la tabla agregando esas columnas (fecha_Nac y edad), después se hace una consulta que nos permite obtener a partir del rfc_Emp obtenido del archivo .csv de cada registros las columnas mencionadas anteriormente:

```

--Para cuando ingresamos registros desde csv, es necesario lo siguiente
ALTER TABLE EMPLEADO ADD fecha_Nac date;--CAMPO CALCULADO
ALTER TABLE EMPLEADO ADD edad smallint;--CAMPO CALCULADO
UPDATE EMPLEADO SET fecha_Nac = CAST(substring (rfc_Emp, 5,6) AS date) WHERE fecha_Nac is NULL;
UPDATE EMPLEADO SET edad = EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM fecha_Nac) WHERE edad is NULL;

```

Figura 37: Obtención de los atributos fecha_Nac y Edad de los registros del leídos del .csv

CatalogoEstados.csv

id_estado	[PK] smallint	nombre_estado	character varying (30)
1		Aguascalientes	
2		Baja California	
3		Baja California Sur	
4		Campeche	
5		Coahuila	
6		Colima	
7		Chiapas	
8		Chihuahua	
9		CDMX	
10		Durango	
11		Estado de México	
12		Guanajuato	
13		Guerrero	
14		Hidalgo	
15		Jalisco	
16		Michoacán	
17		Morelos	
18		Nayarit	
19		Nuevo León	
20		Oaxaca	
21		Puebla	
22		Querétaro	
23		Quintana Roo	
24		San Luis Potosí	
25		Sinaloa	
26		Sonora	
27		Tabasco	
28		Tamaulipas	
29		Tlaxcala	
30		Veracruz	
31		Yucatán	
32		Zacatecas	

Figura 38: CatalogoEstados.CSV

telefonosEmpleados.csv

telefono	[PK] bigint	rfc_emp	character varying (13)
5569674246		OIRM010308	
5558263784		EICG010324F56	
5527363145		BOCC010818	
5566992229		GAOM0108122E9	
5578609654		CAMP0102082V2	
5516543788		OIRD9502265T5	
5543520317		OIGV930114	
5548762672		GOSI010519	
5567033980		SARM991130	
5580600125		RIMA700118	

Figura 39: CatalogoEstados.CSV

meseros.csv

rfc_emp [PK] character varying (13)	horario_dia character varying (5)	horario_hora character varying (5)
BIGV930114	L-J	12-21
GOS1010519	L-J	12-21
SARM991130	J-D	12-21
SOGA031128	J-D	12-21

Figura 40: meseros.csv

EspecialidadCocineros.csv

id_especialidad [PK] smallint	especialidad character varying (50)
1	Parrilla
2	Trompo
3	Salsas
4	Bar
5	Cocina

Figura 41: EspecialidadCocineros.CSV

cocineros.csv

rfc_emp [PK] character varying (13)	id_especialidad [PK] smallint
CAMP0102082V2	3
CAMP0102082V2	5
OIRD950226STS	4
RIMA700118	5
AIRD940610	1
AIRD940610	2
ROMJ940404	1
ROMJ940404	2

Figura 42: especialidadCocineros.CSV

RolAdmin.csv

id_rol [PK] smallint	rol character varying (50)
1	Administrador
2	Gerente
3	Reclutador
4	Proveedor
5	Jefe de meseros
6	Jefe de bar
7	Jefe de cocina
8	Jefe de taqueros

Figura 43: RolAdmin.CSV

Administradores.csv

rfc_emp	[PK] character varying (13)	id_rol	[PK] smallint
OIRM010308	1		
OIRM010308	8		
EICG010324F56	2		
EICG010324F56	5		
BOCC010818	3		
BOCC010818	6		
GAOM0108122E9	4		
GAOM0108122E9	7		

Figura 44: administradores.CSV

Dependientes.csv

curp	[PK] character varying (18)	rfc_emp	[PK] character varying (13)	nombre	character varying (50)	ap_paterno	character varying (50)	ap_materno	character varying (50)	parentesco	character varying (30)
OIRD940226MDFRVN03		OIRM010308	1	Diana	Ortíz	Rivera	Hermana				
OISL171223MDFRRYAS		OIRM010308	8	Layla	Ortíz	Servín	Hija				
GAOE110711HDFLLRA9		GAOM0108122E9	2	Erick	Gabriel	Galán	Olivares	Hermano			
GAOT170618MDFLLNA6		GAOM0108122E9	5	Ingrid	Abigail	Galán	Olivares	Hermana			
EICM040529MDFSRRAB		EICG010324F56	3	Marisol	Selene	Espinosa	Cortez	Hermana			
EICA031030MDFESRLA2		EICG010324F56	6	Ailin	Iratze	Espinosa	Cortez	Hermana			
CAMA960610HDFSNL05		CAMP0102082V2	4	Alan	Castillo	Montes	Hermano				
BOMZ220311MDFRNYAS		BOCC010818	7	Zyanya	Borboa	Mondragón	Hija				
OIRM010308HDFRVGAS		RIMA700118	9	Miguel	Angel	Ortíz	Rivera	Hijo			

Figura 45: Dependientes.CSV

4.3.2. Mediante script

Por otro lado, se realizaron inserciones directamente dentro del manejador, para tanto comprobar el guardado de los datos como insertar datos fijos (como es el caso de la tabla Productos).

Dentro de ellos encontramos:

Nuevos Empleados

Donde:

Values(rfc, num_Emp, nombre, ap_Paterno, ap_Materno, cp, colonia, cale, numero, id_Estado, sueldo, foto)

```
INSERT INTO EMPLEADO VALUES('SOGA031128',NEXTVAL('cuenta_num_emp'),'Andrea','Sosa','Gómez',03650,'Letran Valle','Matías Romero',2016,9,4500,'C:/Users/Miguel/Downloads/Logo de la Universidad de Guadalajara (1).png')
INSERT INTO EMPLEADO VALUES('AIRD940610',NEXTVAL('cuenta_num_emp'),'Daniel','Avila','Rivera',03241,'Roma','Londres',2456,9,8000,'C:/Users/Miguel/Downloads/Logo de la Universidad de Guadalajara (1).png')
INSERT INTO EMPLEADO VALUES('ROMJ940404',NEXTVAL('cuenta_num_emp'),'Julian','Rosas','Monroy',01381,'Juárez','León Calvallo',4590,9,8000,'C:/Users/Miguel/Downloads/Logo de la Universidad de Guadalajara (1).png')
```

Figura 46: Insert Empleados

Telefonos

Donde:

Values(telefono, rfc_Emp)

```

INSERT INTO TELEFONO VALUES(5517699856,'SOGA031128');
INSERT INTO TELEFONO VALUES(5531050789,'AIRD940610');
INSERT INTO TELEFONO VALUES(5544332211,'ROMJ940404');

```

Figura 47: Insert Telefonos

*Ci*entes

Donde:

Values(rfc, nombre, ap_Paterno, ap_Materno, fecha_Nac, razon_social, email, cp, colonia, calle, numero, id_Estado)

```

INSERT INTO CLIENTE VALUES('OIGC700830','Carlos Ernesto','Ortíz','García',NULL,'Carlos Ernesto Ortiz García','carlos_6508@yahoo.com',01280,'Any');
INSERT INTO CLIENTE VALUES('BOCD031020','Daniel','Borboa','Castillo',NULL,'Daniel's Borboas S.A de C.V','danielito_123@hotmail.com',01670,'Alar');
INSERT INTO CLIENTE VALUES('GAOM010812','Miguel Angel','Galán','Olivares',NULL,'Galane's S.A de C.V','galan_123@hotmail.com',09840,'Los reyes');

```

Figura 48: Insert Clientes

*C*ategorias de productos

Donde:

Values(id_Categoría, nom_Categoría, desc_Categoría)

```

INSERT INTO CATEGORIA_PRODUC VALUES(1,'Entradas','Exquisitas entradas para compartir entre amigos y familiares');
INSERT INTO CATEGORIA_PRODUC VALUES(2,'Al pastor','Tacos y gringas');
INSERT INTO CATEGORIA_PRODUC VALUES(3,'Alambres','Carne de cerdo o res sazonada y servida con una porción de queso derretido');
INSERT INTO CATEGORIA_PRODUC VALUES(4,'A la parrilla','Tacos de bistec, rajas, costilla, arrachera y chorizo a la parrilla');
INSERT INTO CATEGORIA_PRODUC VALUES(5,'Sopas','Frijoles charros y sopa de tortilla');
INSERT INTO CATEGORIA_PRODUC VALUES(6,'Bebidas','Refrescos, aguas y jugos');
INSERT INTO CATEGORIA_PRODUC VALUES(7,'Bar','Bebidas alcohólicas');
INSERT INTO CATEGORIA_PRODUC VALUES(8,'Postres','Deliciosos platillos dulces para cerrar con broche de oro');
INSERT INTO CATEGORIA_PRODUC VALUES(9,'Extras','Guarniciones extra de salsa verde o roja, guacamole, chicharrón, etc');

```

Figura 49: Insert Categorias de productos

*P*roductos

Donde:

Values(id_PB, nombre_PB, precio, disponibilidad, descripcion, receta, id_Categoría, es_Platillo, es_Bebida)

```

-----Entradas-----
INSERT INTO PRODUCTO VALUES(1001,'Chicharrón de queso',76,TRUE,'Chicarrón de queso gouda. Salsa a elegir','80 g de queso gouda tostado al comal');
INSERT INTO PRODUCTO VALUES(1002,'Guacamole',63,TRUE,'Guacamole al centro con totopos alrededor','120 g de pulpa de aguacate con cilantro y celi');
INSERT INTO PRODUCTO VALUES(1003,'Cebollitas normales',63,TRUE,'Cebollitas asadas a la plancha','300 g de cebolla cambray',1,TRUE,FALSE);
-----Al pastor-----
INSERT INTO PRODUCTO VALUES(2001,'Gringa de pastor',111,TRUE,'3 Gringas de carne al pastor en tortillas de harina queso, piña, cebolla y cilan');
INSERT INTO PRODUCTO VALUES(2002,'Taco al pastor',15,TRUE,'1 Taco de carne al pastor con piña cebolla y cilantro','22 g de carne al pastor coc');
-----Alambres-----
INSERT INTO PRODUCTO VALUES(3001,'Alambre de pastor',100,TRUE,'Pastor, cebolla, tocino, chile poblano, picado y gratinado con queso manchego',);
INSERT INTO PRODUCTO VALUES(3002,'Alambre de costilla',120,TRUE,'Costilla, cebolla, tocino, chile poblano, picado y gratinado con queso manchego');
INSERT INTO PRODUCTO VALUES(3003,'Alambre de bistec',100,TRUE,'Bistec, cebolla, tocino, chile poblano, picado y gratinado con queso manchego');
INSERT INTO PRODUCTO VALUES(3004,'Alambre vegetariano',90,TRUE,'Alambre de nopales o champiñones a elegir, cebolla, chile poblano, picado y gr');
INSERT INTO PRODUCTO VALUES(3005,'Trinity',130,TRUE,'3 Tacos árabes con alambre de pastor y queso','250 g de pastor, 3 tortillas árabes, 1/4 c');

```

```

-----A la parrilla-----
INSERT INTO PRODUCTO VALUES(4001,'Taco de bistec',15,TRUE,'1 Taco de bistec','22 g de bistec de res cocinado a la parrilla, 2 tortillas de maíz
INSERT INTO PRODUCTO VALUES(4002,'Taco de rajas con crema',12,TRUE,'1 Taco de rajas con crema','30 g de rajas preparadas con crema, 2 tortillas
INSERT INTO PRODUCTO VALUES(4003,'Taco de costilla',20,TRUE,'1 Taco de costilla','30 g de costilla de res cocinada a la parrilla, 2 tortillas c
INSERT INTO PRODUCTO VALUES(4004,'Taco de arrachera',30,TRUE,'1 Taco de arrachera','30 g de arrachera cocinada a la parrilla, 2 tortillas de maíz
INSERT INTO PRODUCTO VALUES(4005,'Taco de chorizo',15,TRUE,'1 Taco de chorizo','25 g de chorizo cocinado a la parrilla, 2 tortillas de maíz de

-----Sopas-----
INSERT INTO PRODUCTO VALUES(5005,'Frijoles charros',60,TRUE,'Plato de frijoles charros','200 g de frijoles cocinados al estilo de la casa en su
INSERT INTO PRODUCTO VALUES(5006,'Sopa de tortilla',60,TRUE,'Plato de sopa de tortilla o azteca','150 g de tortilla dorada en tiras servidas en

-----Bebidas-----
INSERT INTO PRODUCTO VALUES(6006,'Refresco',25,TRUE,'Refrescos: coca-cola, sprite, manzanita, fanta','1 Refresco de envase de vidrio',6,FALSE,
INSERT INTO PRODUCTO VALUES(6007,'Aguas de sabor',25,TRUE,'Aguas de sabor: sandía, limón, jamaica, horchata','1 Agua de sabor servicio
INSERT INTO PRODUCTO VALUES(6008,'Jugos de fruta natural',25,TRUE,'Jugo de fruta natural: naranja, mandarina, verde, zanahoria','1 Jugo de frut

-----Bar-----
INSERT INTO PRODUCTO VALUES(7001,'Cerveza',40,TRUE,'Cerveza: Corona, Modelo, Indio, Sol, Heineken. Oscuras y claras','Cervezas de 355 ml aprox
INSERT INTO PRODUCTO VALUES(7002,'Michelada',50,TRUE,'Michelada: Corona, Modelo, Indio, Sol, Heineken. Oscura y clara','Michelada de 400 ml aprox

-----Postres-----
INSERT INTO PRODUCTO VALUES(8001,'Flan napolitano',30,TRUE,'1 Flan napolitano','Flan preparado a base de huevo, mantequilla y azúcar carameliz
INSERT INTO PRODUCTO VALUES(8002,'Arroz con leche',30,TRUE,'1 Arroz con leche en vaso de plástico','250 ml de arroz preparado con leche, azúcar

-----Extras-----
INSERT INTO PRODUCTO VALUES(9001,'Extra salsa verde o roja',10,TRUE,'Salsa verde o roja','200 ml de salsa verde o roja',9,TRUE,FALSE);
INSERT INTO PRODUCTO VALUES(9002,'Extra de guacamole',20,TRUE,'Guacamole','150 ml de guacamole',9,TRUE,FALSE);
INSERT INTO PRODUCTO VALUES(9003,'Extra de chicharrón',15,TRUE,'Chicharrón','150 g chicharrón',9,TRUE,FALSE);

```

Figura 50: Insert Productos

4.4. Índice

Se decidió realizar un índice de tipo Cluster para los atributos "id_pb, nombre_pb, precio" de la tabla PRODUCTO, ya que como equipo consideramos este conjunto de atributos simulan un menú y siguiendo la descripción de problema, al ser un restaurante, una de las consultas que se realiza con mayor frecuencia es la visualización del menú.

```
CREATE INDEX ind_tab_produc ON PRODUCTO (id_pb, nombre_pb, precio);
```

Figura 51: Indice ind.tab_produc

5. Presentación

Descripción de lo que hace la modalidad seleccionada como forma de conexión hacia la base de datos.

5.1. Interfaz Gráfica

5.1.1. Conexión con la BD

Para la implementación de la interfaz gráfica de este proyecto se optó por utilizar el IDE para desarrollo en Windows Visual Studio 2019, de Microsoft. Descargable desde el sitio oficial de Microsoft.

La elección de este IDE se hizo a partir de la experiencia de los integrantes del equipo, pues previamente ya se habían creado interfaces gráficas para otros proyectos con esta herramienta.

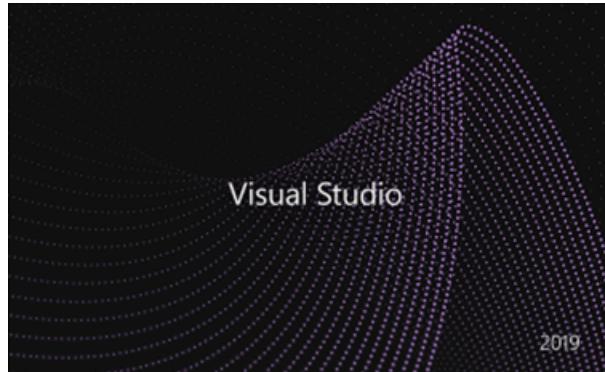


Figura 52: Visual Studio

Lo primero que se hizo fue hacer la conexión del proyecto en Visual Studio con la base de datos en PostgreSQL, para esto, se implementó una estructura Cliente-Servidor, de tal manera que la base de datos está ubicada en una computadora A, que trabaja como el servidor; y la interfaz está ubicada en una computadora B, que trabaja como el cliente.

Para esto, primero instalamos el driver de bases de datos Npgsql desde el Stack Builder de PostgreSQL en la computadora cliente:

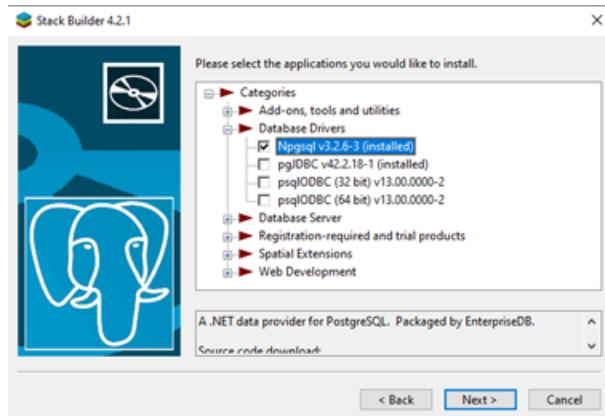
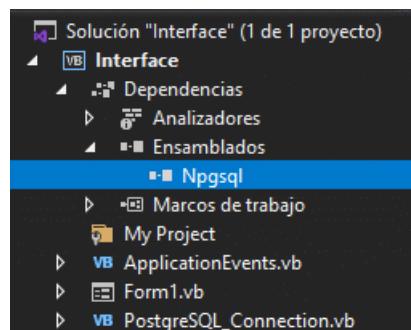


Figura 53: Stack Builder

Una vez que tenemos instalada la biblioteca, la incluimos en el proyecto como Referencia e importamos el driver:



Imports Npgsql

Figura 54: Driver

Ya que tenemos el driver en la interfaz, procedemos a codificar las instrucciones de conexión a la base de datos, y para esto necesitamos conocer la IPv4 de la computadora servidor, dicha IP, la conocemos ingresando el comando ipconfig dentro de la terminal

Figura 55: ipconfig

IP servidor: 192.168.100.114 Además de esto, tenemos que conocer el nombre de la base de datos a la que queremos conectar la interfaz. En este caso, nuestra base de datos tiene por nombre *proyecto_final_postgres*.

Y por último, también necesitamos conocer el usuario y contraseña con la que accederemos a la base de datos en el servidor.

```
Dim host As String = "Host=192.168.100.114;"  
Dim port As String = "Port=5432;"  
Dim db As String = "Database=proyecto_final_postgres;"  
Dim user As String = "Username=postgres;"  
Dim pass As String = "Password=10022001;"  
Dim conString As String = String.Format("{0}{1}{2}{3}{4}", host, port, db, user, pass)
```

Figura 56: Acceso a la BD

En la imagen anterior generamos la cadena *conString* con toda la información necesaria para conectarnos a la base de datos, dicha conexión se hace con el adaptador NpgsqlDataAdapter, el cual es útil para los comandos SELECT, UPDATE, INSERT y DELETE.

Además, inicializamos una instancia de la clase DataTable, que nos servirá para obtener los datos de cada instrucción realizada.

```

Public Function PerformCRUD(Com As NpgsqlCommand) As DataTable
    Dim da As NpgsqlDataAdapter
    Dim dt As New DataTable()
    Try
        da = New NpgsqlDataAdapter
        da.SelectCommand = Com
        da.Fill(dt)
    Catch ex As Exception
        MessageBox.Show("Conexión fallida con la base de datos" & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        dt = Nothing
    End Try
    Return dt
End Function

```

Figura 57: DataTable

Una vez terminada la configuración en la interfaz (cliente), configuraremos la computadora Servidor.

Lo primero que tenemos que hacer es agregar la dirección IP de la computadora cliente dentro del archivo pg_hba.config, el cual se ubica en la carpeta C:/Program Files/PostgreSQL/14/data:

C:\Program Files\PostgreSQL\14\data			
	Nombre	Fecha de modificación	Tipo
do	base	22/05/2022 12:17 a. m.	Carpeta de archivos
Personal	global	26/05/2022 12:22 a. m.	Carpeta de archivos
.	log	26/05/2022 12:21 a. m.	Carpeta de archivos
:	pg_commit_ts	15/02/2022 06:18 p. m.	Carpeta de archivos
tos	pg_dynshmem	15/02/2022 06:18 p. m.	Carpeta de archivos
.	pg_logical	26/05/2022 12:26 a. m.	Carpeta de archivos
dos	pg_multixact	15/02/2022 06:18 p. m.	Carpeta de archivos
D	pg_notify	15/02/2022 06:18 p. m.	Carpeta de archivos
il (C:)	pg_replslot	15/02/2022 06:18 p. m.	Carpeta de archivos
l (C:)	pg_serial	15/02/2022 06:18 p. m.	Carpeta de archivos
l (C:)	pg_snapshots	15/02/2022 06:18 p. m.	Carpeta de archivos
l (C:)	pg_stat	26/05/2022 12:21 a. m.	Carpeta de archivos
l (C:)	pg_stat_tmp	27/05/2022 04:49 p. m.	Carpeta de archivos
l (C:)	pg_subtrans	15/02/2022 06:18 p. m.	Carpeta de archivos
l (C:)	pg_tblspc	15/02/2022 06:18 p. m.	Carpeta de archivos
l (C:)	pg_twophase	15/02/2022 06:18 p. m.	Carpeta de archivos
l (C:)	pg_wal	25/05/2022 10:32 p. m.	Carpeta de archivos
l (C:)	pg_xact	15/02/2022 06:18 p. m.	Carpeta de archivos
l (C:)	current_logfiles	26/05/2022 12:21 a. m.	Archivo
l (C:)	pg_hba	18/05/2022 12:57 a. m.	Archivo CONF
l (C:)	pg_ident	15/02/2022 06:18 p. m.	Archivo CONF

Figura 58: Ruta

Esto se hace para que la computadora cliente pueda hacer peticiones a la computadora servidor.

Dentro del archivo nos vamos hasta la parte de abajo, y en la sección IPv4 local connections agregamos la IPv4 de la computadora cliente (192.168.100.112) y cambiamos los métodos por trust.

```

# TYPE DATABASE USER ADDRESS METHOD

# "local" is for Unix domain socket connections only
local all all trust
# IPv4 local connections:
host all all 127.0.0.1/32 trust
host all all 192.168.100.112/32 trust
# IPv6 local connections:
host all all ::1/128 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all trust
host replication all 127.0.0.1/32 trust
host replication all ::1/128 trust

```

Figura 59: Metodo Trust

Una vez hecho esto, procedemos a hacer el siguiente cambio. El cual consiste en agregar a las variables de entorno la carpeta C:/Program Files/PostgreSQL/14/bin

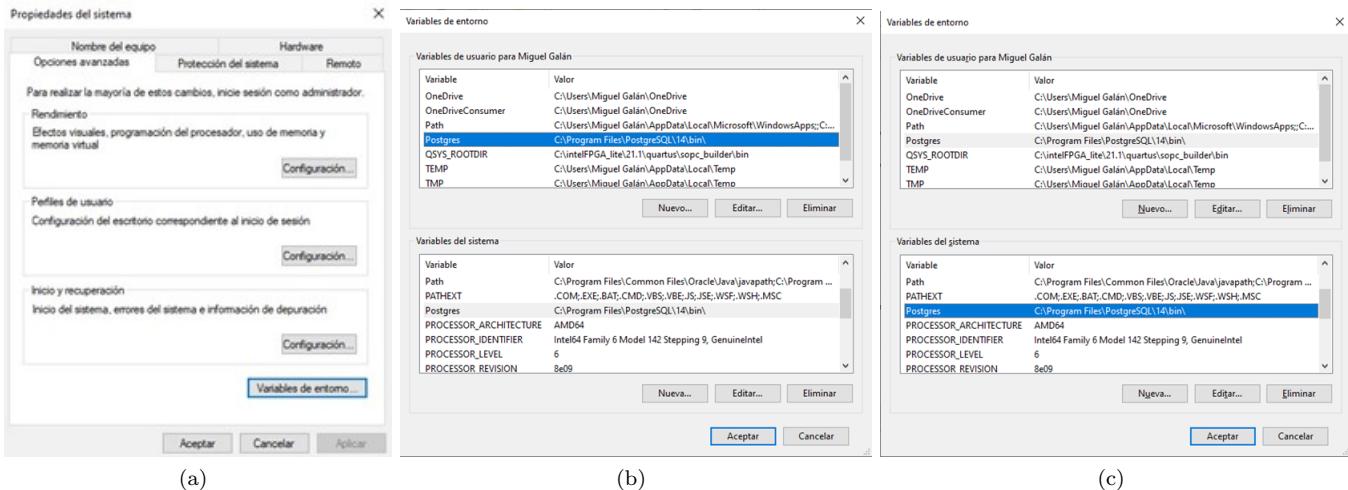


Figura 60: Variables de Entorno

Finalmente guardamos todo y reiniciamos el equipo para que se apliquen los cambios correctamente.

5.1.2. Interfaz

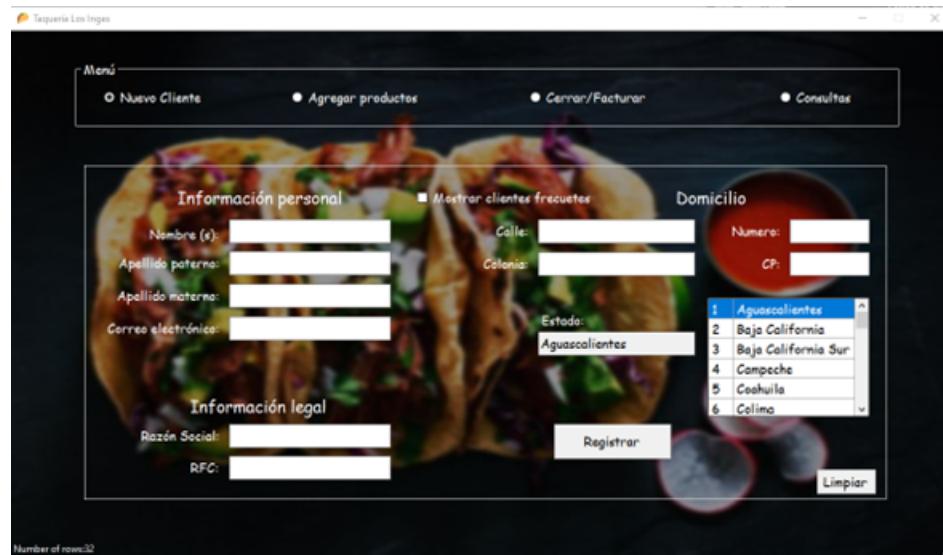


Figura 61: Interfaz Principal

La interfaz cuenta con un menú de 4 secciones: Nuevo Cliente, Agregar productos, Cerrar/Facturar y Consultas. Cada una de estas secciones se habilita con un Radio Button.



Figura 62: Menú Consultas

En la primera sección contamos con una serie de TextBox destinados para que el usuario ingrese la información de cada cliente, como lo es su nombre, correo, razón social, RFC y dirección:

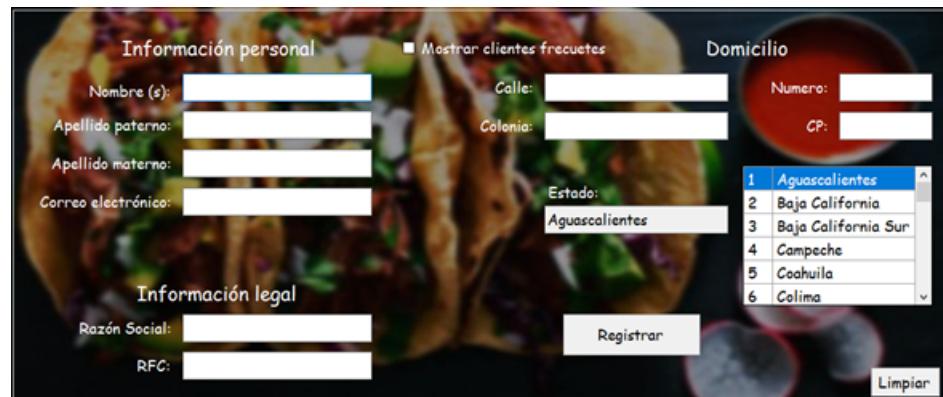


Figura 63: Registrar Nuevo Cliente

Sin embargo, también tenemos la opción de seleccionar a algún cliente que ya fue registrado previamente haciendo click en el CheckBox mostrado en la parte superior:

The screenshot shows a software application window titled "Información personal". At the top right is a checked checkbox labeled "Mostrar clientes frecuentes". Below it is a table with columns: rfc_cliente, nombre, ap_paterno, ap_materno, fecha_nac, razon_social, and email. Two rows of data are visible: one for "Ingrid Galán Olivares" and another for "Nayeli Gómez Esperza". To the right of the table is a section titled "Domicilio" with fields for Calle, Colonia, Numero, and CP. Below these are dropdown menus for "Estado" (with options like Aguascalientes, Baja California, etc.) and "RFC". At the bottom are buttons for "Seleccionar" (Select), "Registrar" (Register), and "Limpiar" (Clear).

Figura 64: Seleccionar cliente ya registrado

Esta opción fue implementada para no tener que ingresar todos los datos del cliente cada que éste visite el restaurante, pues con solo presionar el botón Seleccionar, todos los campos mencionados previamente se llenan de forma automática:

This screenshot shows the same software interface as Figure 64, but with different data entered into the fields. The "Información personal" section now contains: Nombre (s): Ingrid, Apellido paterno: Galán, Apellido materno: Olivares, and Correo electrónico: ingrid_galan@hotmail.com. The "Domicilio" section contains: Calle: Ahuizotl, Colonia: Los Reyes, Numero: 8, and CP: 9840. The "Estado" dropdown menu is open, showing options 1 through 6, with "Aguascalientes" selected. The "Información legal" section contains: Razón Social: Ingrid Galán Olivares, and RFC: GAOI170610. The "Registrar" and "Limpiar" buttons are also present.

Figura 65: Insertar Cliente ya registrado

Posteriormente, al presionar el botón Limpiar, podemos limpiar todos los campos ingresados. Y si presionamos el botón Registrar, pueden pasar dos cosas:

1. Si el cliente ya fue registrado previamente, solo se le crea una nueva cuenta, con un folio diferente al de su(s) cuentas previas.
2. Si el cliente no ha sido registrado, significa que es un nuevo cliente, y por ende, se guardan todos sus datos en la base de datos y posteriormente se abre su primer cuenta.

En la segunda sección tenemos el apartado para agregar productos a cualquier cuenta abierta (el cliente está en el restaurante y puede pedir tantos productos como quiera).



Figura 66: Agregar Producto

En la parte izquierda de dicha sección encontramos la lista de cuentas abiertas, del lado derecho tenemos las listas de platillos y bebidas disponibles y en la parte inferior tenemos la lista desplegable para indicar la cantidad a registrar del producto seleccionado. Una vez seleccionados todos los campos, podemos presionar el botón Agregar producto para registrar el producto en la base de datos y hacer todas las actualizaciones correspondientes.

En la tercera sección tenemos el apartado para cerrar y facturar cuentas:



Figura 67: Cerrar cuenta

Del lado izquierdo tenemos la lista de cuentas abiertas, de las cuales seleccionamos la que queremos cerrar (cuando el cliente paga y se va). Presionamos el botón Cerrar cuenta y en automático se nos despliega la información útil para generar la factura correspondiente en caso de requerirse.

Además, generamos un archivo de texto, que se almacena en la computadora cliente, con la misma información de la factura:

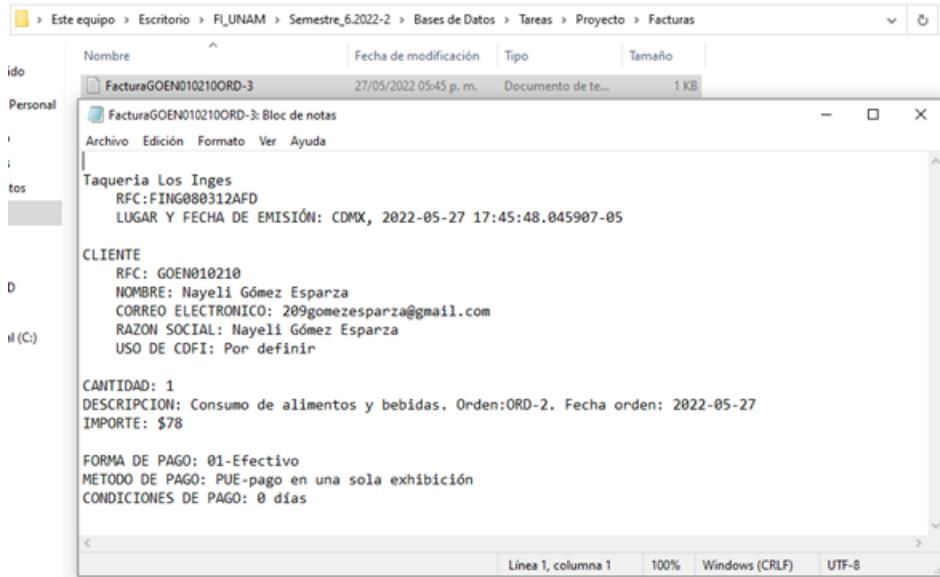


Figura 68: Recibo generado

Por último, en la cuarta sección tenemos otro menú de 5 opciones: Órdenes vendidas por un empleado, Productos más vendidos, Productos no disponibles, Ventas por periodo de tiempo y Empleado.

En la primera opción, tenemos la lista de meseros que trabajan en el restaurante, de los cuales, al seleccionar uno y presionar el botón Buscar, observaremos al centro la cantidad de órdenes (cuentas) atendidas por el mesero seleccionado, junto con la suma total de lo vendido por dicho mesero.

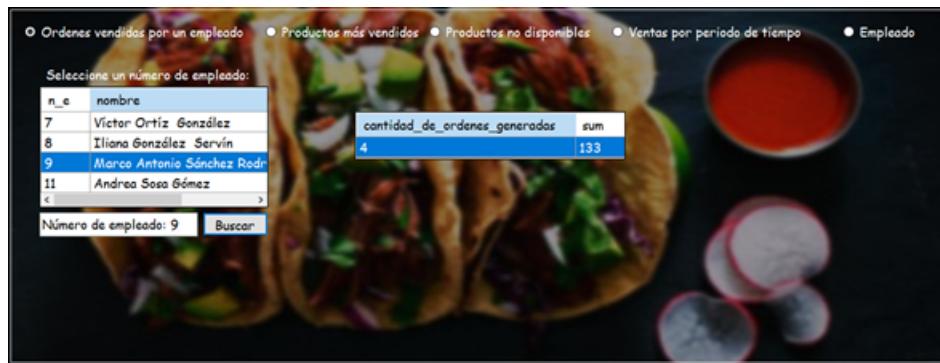


Figura 69: Cantidad de órdenes atendidas

En la segunda opción tenemos los productos más vendidos, mostrando tanto el platillo como la bebida más vendida:

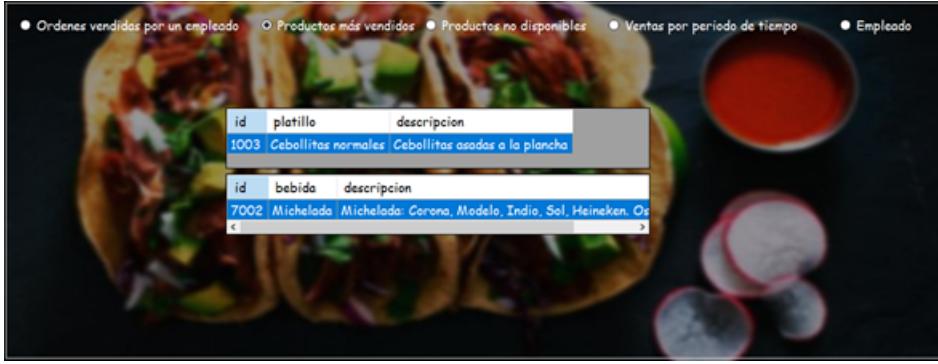


Figura 70: Platillo y bebida más vendido

En la tercera opción tenemos la lista de productos no disponibles, cuando tenemos todos los productos disponibles, observamos un mensaje:

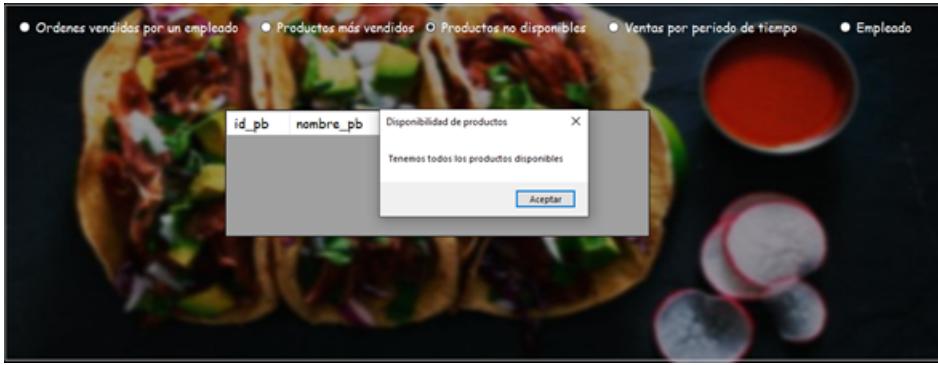


Figura 71: Productos disponibles

En la curta opción podemos hacer la búsqueda de ventas realizadas por día o por periodo de tiempo. Para buscar por día simplemente hay que poner la misma fecha en ambos campos:

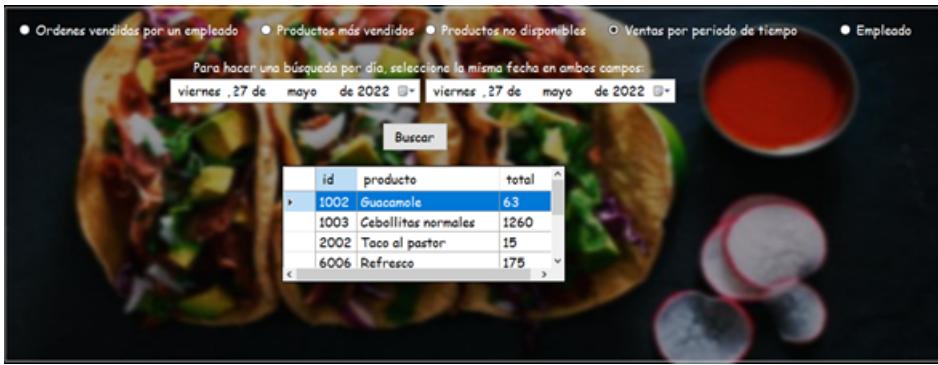


Figura 72: Ventas realizadas en un periodo de tiempo

Por último, en la quinta opción podemos hacer la búsqueda de información de cada uno de los empleados que trabajan en el restaurante:

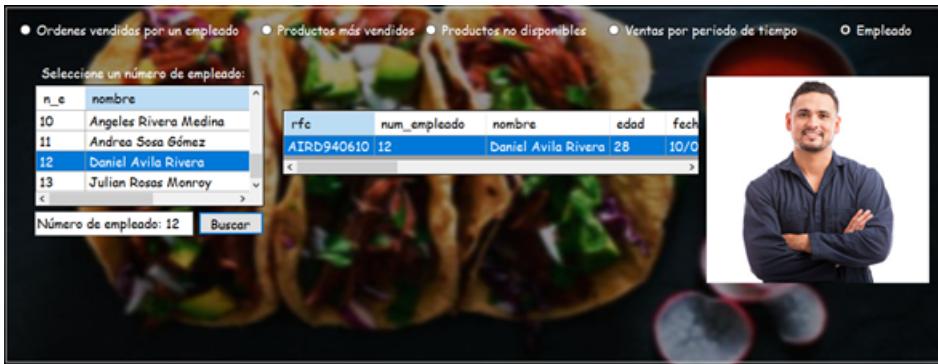


Figura 73: Información de empleados

6. Conclusiones

Personales, detallando las dificultades, retos, aciertos, etc. que se presentaron en el proyecto.

Borboa Castillo Carlos Alfonso

Con base en la realización de este proyecto, se pudieron comprender y poner en práctica los conocimientos teóricos relacionados a las bases de datos. En primera instancia, se llevó a cabo un análisis de requerimientos y, a partir de ellos, se creó un modelo entidad relación aplicando los conceptos y técnicas visualizadas durante las sesiones del curso, adicionalmente, se aplicaron estrategias de diseño más avanzadas para obtener un modelo ER más completo y eficiente. Posteriormente, se obtuvo la representación intermedia y el modelo relacional. Este último permitió hacer uso del software PG Modeler. Finalmente, se generó el modelo físico implementado en el manejador Postgres SQL. En función del resultado obtenido que, corresponde al cumplimiento total de los requerimientos, se concluye que fueron bastantes y correctos los conocimientos adquiridos durante el curso, aunque, es importante mencionar que fue necesario investigar algunas cuestiones particulares acerca de la programación en el manejador.

No obstante, a partir de la creación de la interfaz gráfica y de la elaboración del reporte, se pudo profundizar en la programación en Visual Basic como herramienta para la creación de una aplicación de escritorio, así como el uso del software LATEX como alternativa de programa editor de textos. De manera que, con base en lo anterior, se concluye que la realización del proyecto no solo brindó la posibilidad de aplicar los conocimientos adquiridos en la teoría, también permitió adquirir nuevos conocimientos al utilizar nuevas herramientas que enriquecen la presentación del resultado final.

Castillo Montes Pamela

En lo personal, la realización de este proyecto destaca en varios aspectos, entre ellos la buena organización del equipo, donde consideramos el tiempo necesario para realizar el proyecto y día a día cada integrante fue aportando una parte de su tiempo.

De igual forma, durante el transcurso de su desarrollo gracias a la retroalimentación constante mediante reuniones de meet, nos fue posible realizar cambios y aportar dudas respecto a la mejor manera tanto de diseñar la base de datos como su implementación.

Por otro lado, el no contar con conocimiento suficiente para la creación de una interfaz resultó un reto personal, el cual gracias al conocimiento de mis compañeros no afectó el desarrollo del proyecto.

Espinosa Cortez Giselle

Al leer el enunciado era fácil visualizar las entidades y atributos que nosotros íbamos a implementar en la parte del diseño del MER, tuvimos que batallar un poco con las relaciones con atributos ya que se tenía que pensar de manera más detallada como íbamos a visualizar esa información y que necesitábamos para acceder a ella. Una vez que el MER estuvo aprobado, debo decir que personalmente el este, aunque solo representa la existencia de los

datos deja muy en claro la idea que quieras seguir.

Continuamos realizando la representación intermedia esta causo un poco más de problemas ya mapear de manera correcta porque si una llave foránea no era colocada o era colocada incorrectamente ya no estaríamos representando lo mismo, las llaves foráneas nos ayudan a acceder a la información de otras tablas. Debemos ser muy cuidadosos con la integridad relacional ya que por ejemplo si nosotros queremos referenciar una tabla o registro que no existe pues evidentemente no nos va a dejar, por otra parte, el tipo de dato que debíamos asignar a cada atributo era fundamental, el nombre que le asignábamos a cada atributo era muy importante aquí.

Un error que se cometió en el equipo es que pasamos de la representación intermedia a la implementación de las tablas en el manejador, aunque no perjudico en lo absoluto el diseño se tuvo que pensar en una alternativa de ingeniería inversa para poder obtener el MR, aunque en PGmodeler no pudimos lograr esta tarea y se tuvo que hacer manualmente, en PGAdmi si se logró y se decidió usar el modelo obtenido de allí para después de la aplicación de la normalización ya que se crean las nuevas relaciones que nos aseguran la integridad de la información.

Pasando a la implementación en el manejador se tuvo que hacer uso de funciones específicas para obtener el valor de atributos para cada uno de los registros para las funciones que necesitaban que cada que se ejecutara una instrucción DML se realizara una acción antes, después o durante se usaron los triggers junto con la función correspondiente, por otra parte, las funciones que solo eran para obtener resultados fijos solo se definían y se ejecutaban. Las secuencias nos fueron útiles para poder llevar el conteo de inserción de registros de ciertas relaciones.

Conectar la base de datos con la interfaz fue una de los retos de este proyecto ya que se tienen que seguir una serie de pasos, descargar un driver de pgAdmi que me permita que mi base de datos pueda fungir como servidor remoto para la interfaz gráfica que requiere acceder a la información dentro del manejador

Galan Olivares Miguel Angel

Para este proyecto se requirió una organización muy rigurosa del equipo, empezando con el tiempo, seguido de la distribución de tareas. Comenzamos con las actividades de análisis. Partiendo de los requerimientos proporcionados por el profesor, la primera tarea que realizamos fue el diagrama Entidad Relación, y cuando lo tuvimos completado, pudimos seguir con las actividades subsecuentes al diagrama, como lo es la creación de la base de datos en PostgreSQL. En mi caso particular, me encargué de la creación de la interfaz gráfica, por lo que mi trabajo estuvo muy relacionado con PostgreSQL, el script realizado para la base de datos y la documentación necesaria para trabajar con el IDE donde se desarrolló la interfaz (Visual Studio 2019).

Implementé una estructura Cliente-Servidor para simular un sistema que hace solicitudes a una base de datos remota, y tras haber terminado toda la implementación, me quedo con lo importante que es leer la documentación de todas las herramientas que utilicemos, pues para mí, fue sumamente importante no solo conocer PostgreSQL (PG), y Visual Studio 2019 (VS) por separado, sino que además, tuve que investigar en la documentación cómo hacer el enlace desde una aplicación de escritorio en VS con la base de datos en PG de manera local, para que una vez que todo funcione así, poder escalar el proyecto a la estructura Cliente Servidor. Pero, además, antes de poder empezar con esta parte del proyecto, todas las actividades previas se llevaron a cabo de manera muy eficiente, por lo que también resalto la importancia de una buena organización entre los integrantes de un equipo, con relación a las tareas que se tienen que realizar, y el tiempo en el que tienen que estar listas.

Ortiz Rivera Miguel Angel

A lo largo de la elaboración de este proyecto podemos concluir que, gracias a los conocimientos adquiridos en el curso, logramos analizar una serie de requerimientos específicos para implementar un modelo entidad relación, su representación intermedia, modelo relacional y como punto de inflexión, su creación física por medio de un manejador como Postgres SQL. Observamos como fue posible aplicar todos los conceptos vistos en clase, de manera que nos impresionamos por el éxito obtenido, pues jamás habíamos realizado un proyecto cuyo resultado final fuera un producto perfectamente comercializable.

Pusimos a prueba nuestras capacidades para llevar un modelo físico de una base de datos a una interfaz, donde se apreciará de manera más visual y atractiva, el resultado de todo nuestro aprendizaje del curso.

Identificamos debilidades en el área de programación con SQL, pues aún no habíamos visto de manera práctica conceptos como: funciones y disparadores. No obstante, esto no fue un impedimento para lograr nuestro objetivo principal, pues con ayuda de todo el equipo y el maestro, investigamos todo lo necesario para poder seguir adelante. Por otra parte, un aspecto de mucho trabajo fue encontrar ciertas equivalencias entre manejadores, ya que en el laboratorio empleamos distintas herramientas a las utilizadas en el curso teórico; debido a esto, una dificultad muy notable fue la búsqueda de unificar las aplicaciones que son capaces de realizarse en un manejador, pero en otro no, o aquellas cosas que tienen la misma funcionalidad, pero diferente sintaxis entre manejadores.

Logramos reafirmar nuestro conocimiento mediante herramientas como: pgAdmin4, pgModeler, Visual Basic y diagrams.net, con los cuales pusimos en práctica cada una de las etapas para realizar la construcción de una correcta base de datos.