

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA
BASES DE DATOS

Proyecto Final

Semestre 2022-2

Equipo 7
Abrego Abascal Diego
Hernández Rojas Mara Alexandra
Rosales López Luis André
Vargas Pacheco Bryan

Profesor
Ing. Fernando Arreola

28 de mayo de 2022

Índice

1. Introducción	1
1.1. Objetivo del proyecto	1
1.2. Análisis del Problema	1
1.3. Propuesta de solución	1
2. Plan de Trabajo	1
2.1. Actividades a realizar	1
2.2. Plan de actividades	1
2.3. ¿Qué hizo cada miembro del equipo?	2
3. Diseño	2
3.1. Modelo Entidad Relación Extendido (MERE)	2
3.2. Modelo Relacional (MR)	3
3.3. Normalización	4
4. Implementación	4
4.1. Lenguaje de definición de datos DDL	4
5. Presentación	10
5.1. ¿Cómo utilizar la interfaz?	10
6. Conclusiones	14
6.1. Abrego Abascal Diego	14
6.2. Hernández Rojas Mara Alexandra	15
6.3. Rosales López Luis André	15
6.4. Vargas Pacheco Bryan	16

1. Introducción

1.1. Objetivo del proyecto

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

1.2. Análisis del Problema

Se requiere desarrollar un sistema informático para administrar un restaurante, su objetivo principal será el almacenamiento de información relevante respecto a los empleados, los productos a la venta, los clientes y las ventas que se realizan (órdenes).

En el caso de los empleados estos pueden ser de tipo administrativo, cocinero y/o mesero, es necesario conocer su RFC, número de empleado, nombre, fecha de nacimiento, teléfonos, edad, domicilio, sueldo, fotografía y dependiendo el caso horario, rol o especialidad, así como la información personal de sus dependientes (CURP, nombre y parentesco).

Para las órdenes contaremos con un folio que nos permita identificarla, toda la información referente a su contenido, el mesero que la atendió y el cliente que la ordenó (RFC, nombre, domicilio, razón social, e-mail y fecha de nacimiento) en caso de que solicite una factura.

Por otro lado, los productos que se vendan deben contar con una descripción, nombre, receta, precio y indicador de disponibilidad, además de indicar si califican como platillos o como bebidas.

1.3. Propuesta de solución

Al analizar el problema se identificó que era posible establecer una relación de especialización partiendo del empleado, hacia los diferentes puestos (los más especializados) como son el de cocinero, administrativo y mesero elegimos hacer uso del Modelo Entidad Relación Extendido (MERE) en vez del Modelo Entidad Relación (MER). También proponemos crear una aplicación web para visualizar nuestra interfaz gráfica debido a que los miembros de nuestro equipo cuentan con mayor experiencia en el desarrollo web que en el desarrollo móvil.

2. Plan de Trabajo

2.1. Actividades a realizar

Como producto final se debe entregar la interfaz gráfica del sistema informático por medio de una aplicación móvil o una aplicación web así como la documentación que justifique la construcción del sistema comenzando por el proceso de análisis, siguiendo con el diseño del modelo entidad-relación, proceso de normalización, definición de tablas en lenguaje DDL, creación de tablas, uso de funciones, triggers y vistas, traslado a aplicación web más instrucciones de uso.

2.2. Plan de actividades

Se pretende avanzar con el proyecto en medida de lo posible durante los días laborales y dedicarle un horario establecido los fines de semana.

2.3. ¿Qué hizo cada miembro del equipo?

El análisis del problema y propuesta de solución fue responsabilidad de los cuatro miembros del equipo para asegurarnos de que durante el desarrollo no nos desviáramos del diseño, de igual forma la redacción de este documento y la elaboración de las diapositivas para la exposición fueron trabajadas en conjunto con las herramientas Overleaf y Google Drive.

La primera parte del trabajo será trabajada principalmente por Luis André y Bryan mientras que la segunda la desarrollan Diego y Mara, por esta razón es muy importante mantener una comunicación efectiva por medio de un canal en Discord y un grupo de WhatsApp.

3. Diseño

3.1. Modelo Entidad Relación Extendido (MERE)

En el desarrollo del Modelo Entidad Relación Extendido se identificaron las siguientes entidades: EMPLEADO, ADMINISTRATIVO, COCINERO, MESERO, DEPENDIENTE, ORDEN, PRODUCTO, CLIENTE. A su vez se detectó que la especialización del empleado hacia los diversos puestos presentaba una relación de traslape y de completitud parcial dado que un empleado puede ocupar varios puestos. Se colocaron los diferentes atributos necesarios y finalmente se identificaron las siguientes relaciones. Las entidades serán colocadas en mayúsculas mientras que el nombre de las relaciones sera indicado con minúsculas seguido de la cardinalidad de la misma.

1. DEPENDIENTE(obligatorio) depende(M:1) EMPLEADO(opcional)
2. COCINERO(opcional) prepara(1:M) PRODUCTO(obligatorio)
3. MESERO(obligatorio) atiende(1:M) ORDEN(opcional)
4. ORDEN(obligatorio) realiza(M:1) CLIENTE(obligatorio)
5. PRODUCTO(opcional) pertenece(M:1) CATEGORIA(obligatorio)
6. ORDEN(opcional) contiene(M:1)(atributos: total_por_producto, cantidad) PRODUCTO(obligatorio)

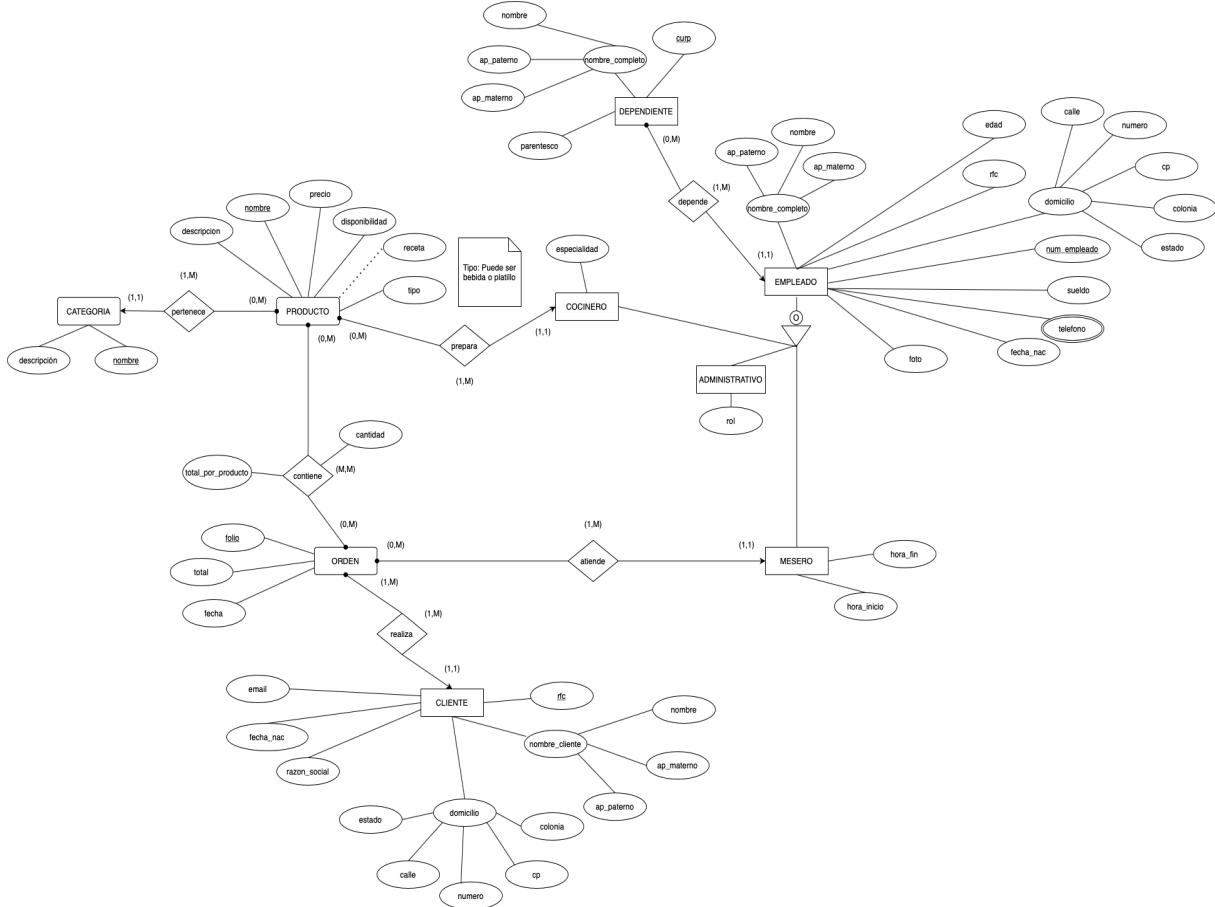


Figura 1: MER

3.2. Modelo Relacional (MR)

Este modelo se realizó con la herramienta PGModeler en base al MER propuesto, cuidando la relación de especialización entre el empleado, mesero, administrativo y cocinero; se puede observar en la figura como esta condición hace que la tabla empleado se relacione directamente a producto y a orden.

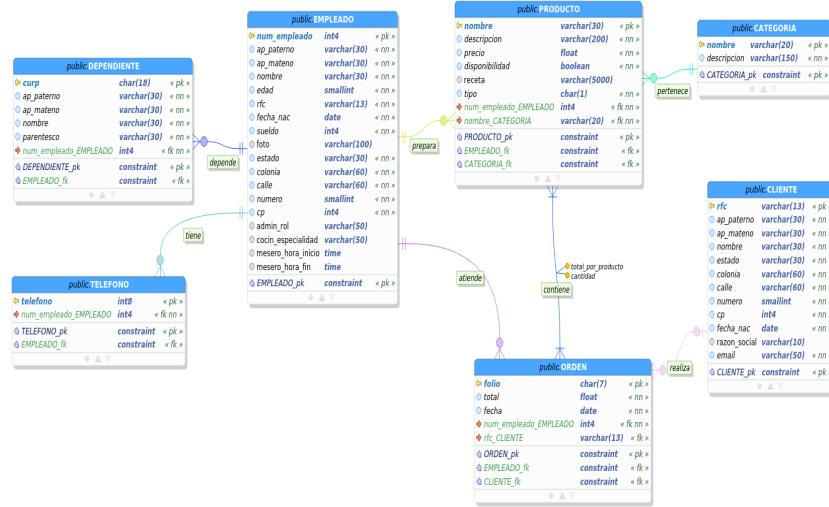


Figura 2: MR

3.3. Normalización

Al llevar a cabo el proceso de normalización sobre las tablas que obtuvimos en el MR pudimos observar que nuestro diseño presentaba un poco de redundancia de datos en la entidad empleado por la misma propuesta de solución (incluir una entidad general y tres específicas) pues en nuestro afán de compactarla dedicando nuestro tiempo al análisis del problema y la reflexión grupal sobre la propuesta de diseño nos dio el tiempo suficiente para invertir en el desarrollo de la aplicación web pero nos dejó sin la posibilidad de normalizar la solución.

4. Implementación

4.1. Lenguaje de definición de datos DDL

Una vez que se realizó el modelo entidad relación extendido de la base de datos, se transcribió a un modelo relacional y finalmente pasó por un proceso de normalización se deben describir los datos y relaciones de la base de datos dentro de un entorno para la creación de la base de datos. Esto se realiza mediante el lenguaje DDL(Data Definition Language) el cual es una de las categorías en las que se clasifica el lenguaje SQL basándose en su uso. El lenguaje DDL nos permite la creación, modificación y eliminación de estructura de objetos de una base de datos. Se muestra la explicación detallada de la implementación del equipo en el lenguaje DDL.

```
CREATE DATABASE proyectoFinal;
```

Figura 3: Se realiza esta sentencia para la creación de la base de datos

Una vez creada la base de datos es necesario utilizar algunas sentencias para la creación de cada una de las tablas, a continuación se muestra la sintaxis necesaria:

- Para la creación de la tabla se usa: **create table nombreTabla**

- Para la creación de los atributos de las tablas se utiliza: nombreAtributo **tipo_de_dato** (tamaño de ser necesario) **NULL/NOT NULL**

Explicado de otra forma se requiere especificar el nombre del atributo, el tipo de dato del atributo, de ser necesario el atributo(Esto solo aplica a tipos de dato como char), y si el valor del atributo es obligatorio o no. Es necesario también hablar de los constraints, mejor conocidos como las reglas para los datos dentro de una tabla.

La sintaxis básica de un constraint es la siguiente: **constraint nombreConstraint tipoConstraint expresión**

Ya con esta información se muestra la implementación realizada para la creación de las tablas dentro de la base de datos:

```
CREATE TABLE categoria(
    nombre varchar(20) NOT NULL,
    descripcion varchar(150) NOT NULL,
    CONSTRAINT categoria_pk PRIMARY KEY (nombre)
);
```

Figura 4: Creación de la tabla categoría

```
CREATE TABLE cliente(
    rfc varchar(13) NOT NULL,
    ap_paterno varchar(30) NOT NULL,
    ap_materno varchar(30) NOT NULL,
    nombre varchar(30) NOT NULL,
    estado varchar(30) NOT NULL,
    colonia varchar(60) NOT NULL,
    calle varchar(60) NOT NULL,
    numero smallint NOT NULL,
    cp int NOT NULL,
    fecha_nac date NOT NULL,
    razon_social varchar(10),
    email varchar(50) NOT NULL,
    CONSTRAINT cliente_pk PRIMARY KEY (rfc)
);
```

Figura 5: Creación de la tabla cliente

```
CREATE TABLE dependiente(
    curp char(18) NOT NULL,
    ap_paterno varchar(30) NOT NULL,
    ap_materno varchar(30) NOT NULL,
    nombre varchar(30) NOT NULL,
    parentesco varchar(30) NOT NULL,
    num_empleado int NOT NULL,
    CONSTRAINT dependiente_pk PRIMARY KEY (curp)
);
```

Figura 6: Creación de la tabla dependiente

```
CREATE TABLE empleado(
    num_empleado int NOT NULL,
    ap_paterno varchar(30) NOT NULL,
    ap_materno varchar(30) NOT NULL,
    nombre varchar(30) NOT NULL,
    edad smallint NOT NULL,
    rfc varchar(13) NOT NULL,
    fecha_nac date NOT NULL,
    sueldo int NOT NULL,
    foto varchar(100),
    estado varchar(30) NOT NULL,
    colonia varchar(60) NOT NULL,
    calle varchar(60) NOT NULL,
    numero smallint NOT NULL,
    cp int NOT NULL,
    admin_rol varchar(50),
    cocin_especialidad varchar(50),
    mesero_hora_inicio time,
    mesero_hora_fin time,
    CONSTRAINT empleado_pk PRIMARY KEY (num_empleado)
);
```

Figura 7: Creación de la tabla empleado

```

CREATE TABLE producto(
    nombre varchar(30) NOT NULL,
    descripcion varchar(200) NOT NULL,
    precio float NOT NULL,
    disponibilidad boolean NOT NULL,
    receta varchar(5000),
    tipo char(1) NOT NULL,
    num_empleado int NOT NULL,
    nombre_categoria varchar(20) NOT NULL,
    CONSTRAINT producto_pk PRIMARY KEY (nombre)
);

```

Figura 8: Creación de la tabla producto

```

CREATE TABLE orden(
    folio char(7) NOT NULL,
    total float NOT NULL,
    fecha date NOT NULL,
    num_empleado int NOT NULL,
    rfc_cliente varchar(13),
    CONSTRAINT orden_pk PRIMARY KEY (folio)
);

```

Figura 9: Creación de la tabla orden

En cuanto a los procedimientos, funciones, triggers y vistas, a continuación se dará una explicación del funcionamiento de cada una de ellas. Cabe recordar que estas nos ayudan a cubrir los requerimientos especiales del proyecto.

La primer función y el primer trigger fueron creados con el propósito de poder formatear el folio de una nueva orden. Este requerimiento establecía que el folio de las ordenes debía tener el formato **ORD-001**. Para lograr esto fue necesario realizar las siguientes acciones:

- Crear una secuencia que nos ayuda a mantener el conteo de registros insertados, de tal forma que los registros sean consecutivos.
- Crear una función que nos ayude a generar el nuevo folio. Esto se logra al modificar el valor del atributo *folio* utilizado en la query de inserción. Además, se utiliza la función *concat()* para unir el prefijo con

el número de secuencia que corresponde. Finalmente se utiliza la función `to_char()` para convertir el tipo de dato de entero a caracteres y hacerlo seguir el formato '000', de tal forma que la cadena quede como **ORD-001** y no como **ORD-1**.

- Crear un trigger que ejecute la función definida con anterioridad antes de que se realice una inserción en la tabla `orden`, de tal forma que sea posible manipular el valor de los atributos pasados en la query.

```

CREATE SEQUENCE seq_id_orden AS integer START 1;
CREATE OR REPLACE FUNCTION formateo_folio_orden()
RETURNS trigger AS $$
BEGIN
    NEW.folio = concat('ORD-', to_char(nextval('seq_id_orden'), 'fm000'));
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER formateo_folio_orden
BEFORE INSERT ON orden
FOR EACH ROW
    EXECUTE PROCEDURE formateo_folio_orden();

```

Figura 10: Función y trigger que formatean el folio de una nueva orden

La segunda función y trigger que creamos tienen el propósito de realizar la actualización de los totales (por producto y venta) cada que se agregue un producto a la orden, así como validar que el producto esté disponible. Determinamos que lo ideal para el trigger es que este se active en operaciones de inserción o actualización, pues en el caso de nuestra implementación estamos forzando el tener solo un registro por orden por cada tipo de producto.

En otras palabras, si ya existe un producto en la orden y el cliente decide ordenar más del mismo tipo, estos se suman al registro ya existente.

```

CREATE OR REPLACE FUNCTION actualizacion_orden()
RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'INSERT' AND EXISTS(select 1 from productos_orden where nombre_producto = NEW.nombre_producto AND folio_orden =
    NEW.folio_orden) = true THEN
        RAISE EXCEPTION SQLSTATE '90001' USING MESSAGE = 'Ese producto ya se encuentra en la orden, actualiza su cantidad';
    ELSE
        IF (SELECT disponibilidad FROM producto WHERE nombre=NEW.nombre_producto) = true THEN
            NEW.total_por_producto = NEW.cantidad * (SELECT precio FROM producto WHERE nombre=NEW.nombre_producto);

            IF EXISTS(SELECT 1 FROM orden WHERE folio=NEW.folio_orden) = true THEN
                UPDATE orden SET total = anterior.total+NEW.total_por_producto FROM (SELECT total FROM orden WHERE folio=NEW.folio_orden) AS
                anterior where folio = NEW.folio_orden;
            ELSE
                RAISE EXCEPTION SQLSTATE '90003' USING MESSAGE ='Esa orden no existe';
            END IF;

        ELSE
            RAISE EXCEPTION SQLSTATE '90002' USING MESSAGE ='El producto ya no se encuentra disponible';
        END IF;
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER actualizar_orden
BEFORE INSERT OR UPDATE ON productos_orden
FOR EACH ROW
EXECUTE PROCEDURE actualizacion_orden()

```

Figura 11: Función y trigger que actualiza los valores de las órdenes, tanto a nivel producto como a nivel total

El siguiente requerimiento que cumplimos es el de la creación de un índice del tipo y posición deseados por nosotros. El índice que decidimos crear es de tipo **HASH** y este lo referenciamos a la columna de *num_empleado* de la tabla *orden*. Decidimos utilizar ese tipo pues identificamos que las comparaciones que haríamos serían de igualdad.

Además, lo referenciamos a dicha columna puesto que uno de los requerimientos nos pedía obtener el número de órdenes y total cobrado por empleado. Esta información se puede obtener toda de la tabla *orden*. Por su naturaleza esta tabla puede llegar a tener un tamaño muy considerable, por lo que consideramos es donde se obtendrían mayores beneficios.

```
create index empleado_orden_index on orden USING HASH(num_empleado);
```

Figura 12: Se elige usar HASH pues solo se harán comparaciones de igualdad

También generamos funciones que no están ligadas a algún trigger. La siguiente función que creamos tiene como objetivo cubrir el requerimiento descrito en el párrafo anterior. Lo destacable de esta función es que utilizamos el parámetro de la misma para modificar nuestra query y obtener los datos del empleado correspondiente. Además, devolvemos el resultado dentro de una tabla. Es el caso más claro que tenemos de uso de las funciones de agregación, también.

```
CREATE OR REPLACE FUNCTION ordenes_por_empleado(n_empleado int)
RETURNS TABLE(num_emp int, total_ordenes bigint, total_cobrado float)
AS $$ 
BEGIN
    RETURN QUERY select num_empleado, count(*), sum(total) from orden where num_empleado = $1 group by num_empleado;
END;
$$
LANGUAGE plpgsql;
```

Figura 13: Función para obtener la información de órdenes por empleado

La siguiente función es muy similar a la anterior. Esta función la creamos para resolver el requerimiento de obtener el total de ventas y total cobrado en un periodo de tiempo establecido. La diferencia en esta función se encuentra en los parámetros de entrada que son de tipo *date* y en que la consulta hace uso del operador *between* para obtener todos los registros que se encuentran dentro del rango de tiempo indicado.

```
CREATE OR REPLACE FUNCTION reporte_parcial(inicio date, fin date)
RETURNS TABLE(ventas_del_periodo bigint, cobro_total_del_periodo float) AS $$ 
BEGIN
    RETURN QUERY SELECT count(*), sum(total) from orden where fecha between inicio and fin;
END;$$
LANGUAGE plpgsql;
```

Figura 14: Función para obtener las ventas y el total cobrado durante un periodo

El requerimiento que costó más trabajo cumplir fue el de generar una vista de forma automática que devuelva la información de manera semejante a como se mostraría en una factura. Nosotros consideramos que los datos que se muestran de forma usual en una facturan son el concepto, la cantidad, el precio y el total por cada tipo de producto. Lo que hizo difícil a esta función fue que no sabíamos como utilizar el parámetro que se pasa a la función dentro de la query para crear la vista. Comenzamos queriendo implementar esta función como un procedimiento, pero postgres solo permite realizar operaciones DML dentro de estos.

```

CREATE OR REPLACE FUNCTION generar_vista_factura(f_orden text)
RETURNS void
AS $$ 
BEGIN
    EXECUTE 'CREATE OR REPLACE VIEW vista_factura AS ' || '
        SELECT po.nombre_producto AS concepto, p.precio, po.cantidad, po.total_por_producto as total FROM
        productos_orden po join producto p on nombre_producto = nombre WHERE folio_orden = ' || quote_literal(f_orden);
END;$$
LANGUAGE plpgsql;

```

Figura 15: Función para generar una vista de forma dinámica

Finalmente, en cuanto a las vistas, la primera que creamos nos permite obtener los detalles del producto más vendido. Como sabemos, la llamada a una vista genera la ejecución de la query ligada a la misma. En este caso hacemos uso de una sub-consulta para poder determinar el nombre del producto más vendido según la tabla *productos_orden* y el resulta utilizarlo para obtener su información de la tabla *producto*.

```

CREATE VIEW producto_mas_vendido AS
    SELECT * FROM producto WHERE nombre = (SELECT nombre_producto FROM productos_orden GROUP BY nombre_producto ORDER BY COUNT(*) DESC LIMIT
    1);

```

Figura 16: Vista para obtener los detalles del producto más vendido

El último requerimiento que resolvimos es aquel que nos pedía poder obtener el nombre de aquellos producto que no estén disponibles. Esto requerimiento lo cumplimos con ayuda de una vista, misma que ejecuta la sentencia correspondiente para obtener el nombre de todos los productos cuya disponibilidad es falsa dentro de la tabla *producto*

```

CREATE VIEW productos_no_disponibles (nombre_producto) AS
    SELECT nombre FROM producto WHERE disponibilidad = false;

```

Figura 17: Vista para obtener los nombres de los productos no disponibles

5. Presentación

5.1. ¿Cómo utilizar la interfaz?

Para interactuar con la interfaz gráfica debe entrar a la siguiente dirección web www.proyectobd.xyz en la pantalla de inicio usted pude seleccionar entre ver los datos de los empleados o llenar una orden.



Figura 18: Pantalla de inicio

Si selecciona el botón de empleados se puede visualizar una tabla con la información general de los empleado, y si se presiona sobre su nombre podemos ver la información faltante correspondiente a su puesto (Especialización).



Figura 19: Tabla con información general



Figura 20: Información específica de un empleado

En el caso de llenar una orden se le presenta al usuario la opción de elegir hasta tres productos, se mostrarán sus precios individuales y se solicitará ingresar la cantidad deseada de dicho producto, para al final mostrar el total. Las órdenes registradas con anterioridad se muestran en una tabla al final de la página.

Producto	Precio	Cantidad	Total
Producto 1 Hot Cakes	\$85	1	\$85
Producto 2 Arachera	\$175	1	\$175
Producto 3 Cafe Americano	\$30	1	\$30
		Total Final	\$290

Procesar Orden

Ordenes Registradas

No se ordenes registradas

Figura 21: Sin órdenes registradas



Llene los datos de la orden

Producto	Precio	Cantidad	Total
Producto 1 Elija un producto	0		
Producto 2 Elija un producto	0		
Producto 3 Elija un producto	0		
Total Final			
Procesar Orden			

Ordenes Registradas

Folio	Fecha	Total	Nombre Empleado	RFC Cliente
ORD-001	2022-05-28	290	Ramirez Ramirez Carlos	MELM8305281H4

Figura 22: Una orden registrada

Llene los datos de la orden

Producto	Precio	Cantidad	Total
Producto 1 Hot Cakes	85	5	425
Producto 2 Hot Cakes	85	3	255
Producto 3 Café Americano	30	3	90
Total Final			770
Procesar Orden			

Ordenes Registradas

Folio	Fecha	Total	Nombre Empleado	RFC Cliente
ORD-001	2022-05-28	290	Ramirez Ramirez Carlos	MELM8305281H4

Figura 23: Registrando segunda orden



Llene los datos de la orden

Producto	Precio	Cantidad	Total
Producto 1: Elija un producto	0		
Producto 2: Elija un producto	0		
Producto 3: Elija un producto	0		
Total Final			

Procesar Orden

Ordenes Registradas

Folio	Fecha	Total	Nombre Empleado	RFC Cliente
ORD-002	2022-05-28	770	Ramirez Ramirez Carlos	MELM830528H4
ORD-001	2022-05-28	200	Ramirez Ramirez Carlos	MELM830528H4

Figura 24: Dos órdenes registradas

6. Conclusiones

6.1. Abrego Abascal Diego

Al realizar este proyecto pusimos en practica absolutamente todos los conceptos aprendidos, si no es que la gran mayoría, tanto de forma practica como teórica a lo largo del semestre. La primera de las dificultades a la que nos enfrentamos fue el realizar el diseño de una forma que nos permitiese cubrir todos los requisitos solicitados mientras el diseño realizado era uno que conservara eficiencia y una estructura adecuada para poder insertar o buscar información según era requerido. El segundo reto al que nos enfrentamos como equipo fue el mantener la organización de tiempo y de trabajo para los diferentes rubros del proyecto, esto en gran parte debido a como se presento el semestre en modalidad mixta.

La siguiente gran dificultad del proyecto se presento a la hora de realizar la programación dentro de la base de datos creando las diferentes funciones y triggers para los eventos solicitados ya que requirió mucha investigación por nuestra parte y se trataba de conceptos en ocasiones bastante abstractos.

Otro de los principales retos y aciertos fue la creación de la interfaz junto con la aplicación web. Se presentaron muchos retos ya que no todos teníamos la misma versión de postgres y el entorno de desarrollo entre nosotros no era exactamente el mismo lo que provocó complicaciones y pasos extras que resultaban confusos.

A pesar de todas las dificultades que se presentaron considero que en general el proyecto fue un acierto ya que desde la etapa de desarrollo pudimos llegar a un modelo que nos permitió satisfacer todos los requisitos solicitados en las etapas posteriores de forma satisfactoria. En conclusión creo que el proyecto definitivamente nos ayudo a cimentar todo lo aprendido a lo largo del semestre cumpliendo exitosamente el objetivo propuesto.

6.2. Hernández Rojas Mara Alexandra

A mi parecer enfrentamos dos grandes retos con la realización de este proyecto, el primero fue lograr pactar una solución definitiva al problema pues en cada escenario que planteábamos existía un inconveniente que nos provocaría problemas en las consultas o en su interacción con los otros elementos del sistema, sin embargo valió la pena el tiempo invertido en la etapa de diseño pues nos permitió llegar a la solución que consideramos más eficiente.

El segundo de los retos fue la implementación web pues esta nos sacaba de nuestra zona de comfort y requería de conocimientos de los cuales no tenemos antecedentes en la facultad y aún así logramos obtener un resultado satisfactorio.

La mayor dificultad que enfrentamos fue el manejo de distintas versiones de Postgres entre los diferentes miembros del equipo pues unos hacían sus manejos desde máquinas virtuales (con diferentes versiones cada uno) y otros teníamos una versión desactualizada instalada en nuestras máquinas. En segundo lugar vendría el manejo de nuestro tiempo por la situación del semestre híbrido no pudimos coincidir tener tantas sesiones grupales como planeamos, por lo cual avanzamos de forma asíncrona.

En cuanto a nuestros aciertos puedo decir que logramos satisfacer el objetivo general al crear un sistema informático que cumple con los requerimientos del problema y que tiene una implementación amigable con el usuario común, al trabajar en equipo pudimos comunicar de manera efectiva nuestras ideas o preocupaciones a nuestros compañeros de equipo y en conjunto buscamos soluciones que nos forzaron a utilizar todos los conocimientos que nos brindó el curso de bases de datos. Por estos motivos considero que el proyecto nos ayudó a desarrollar nuestras habilidades blandas y duras.

6.3. Rosales López Luis André

En este proyecto final fue posible poner en práctica cada uno de los temas vistos durante el semestre. En lo personal, no considero que el proyecto estuviese difícil. No obstante, cada una de las fases requirió un alto nivel de atención a los detalles más pequeños.

Por poner un ejemplo, el planteamiento a MERE de los requerimientos establecidos nos hizo plantear diversas soluciones, cada una con sus pros y sus contras. No logré quedar satisfecho al cien porciento con ninguna de ellas, pero elegimos la que consideramos resolvía mejor el problema.

La parte que me pareció más interesante fue la creación de las funciones, procedimientos, triggers y vistas, pues creo que es maravilloso el poder agrupar operaciones complejas para manipular los datos y tener la posibilidad de ejecutar dichas operaciones con una simple llamada.

Hubo algunos puntos que considero habrían sido más fáciles de implementar fuera del manejador de la base de datos, directo en el back-end, aunque entiendo que implementarlos dentro permite evitar que se realicen ciertas queries externas. Además, creo que en algunos casos sería hasta positivo tener las validaciones en ambas puntas, tanto en el servidor como en el manejador.

Pienso que el mayor reto fue el manejar los tiempos en este semestre que considero, ha sido el más difícil que he tenido en toda la carrera. No creo que sea por las materias, sino por el cambio de ritmo que representó el volver a las clases presenciales. Además, esto es algo que ha afectado a todos en mayor o menor forma, lo que derivó en cierta dificultad para lograr coincidir en horarios con los compañeros.

En conclusión, creo que este proyecto me ayudó a seguir fortaleciendo mi capacidad para trabajar en equipo y me mostró ciertas deficiencias que tengo a la hora de administrar el tiempo, sobre todo por haberme acostumbrado en exceso al trabajo desde casa, en donde definitivamente el tiempo rinde más. Además, pude

fortalecer mis conocimientos de la materia e incluso llenar algunos huecos conceptuales que me quedaron a lo largo del curso.

6.4. Vargas Pacheco Bryan

Considero que la elaboración de este proyecto fue un gran reto, ya que se requirió de implementar la mayoría de los conceptos que se vieron durante el curso.

Para la elaboración se requirieron conocimientos sobre el modelo entidad relación extendido, modelo relacional, normalización, desarrollo web y en general lo relacionado al lenguaje sql en DDL, DML, DCL. En general considero que la parte más complicada del desarrollo de este proyecto derivó en la implementación de la base de datos en posgressql ya que se requirió poner especial atención e ciertas restricciones que el lenguaje sql tiene dentro de este manejador y a partir de ellas pensar en la elaboración de los triggers, vistas, funciones y procedimientos.

En cuanto al diseño considero que hay cosas que pueden ser mejoradas, pero estoy satisfecho con los resultados, ya que desde la perspectiva del equipo se considera que cumple con cada uno de los puntos establecidos durante la elaboración del proyecto.

Además de haber reforzado mis conocimientos sobre bases de datos relacionales, este proyecto me permitió mejorar mis habilidades de comunicación en equipos de trabajo, ya que tuve que trabajar con personas con las que no había tenido la oportunidad de trabajar anteriormente y además cada una de las reuniones se llevó a cabo en línea, por lo que se requirió de aún más habilidades de comunicación.