Universidad Nacional Autónoma de México

Facultad de Ingeniería

Proyecto Final

Integrantes:

- Hernandez Ramirez Miguel Angel
- Rivera López David Zaid
- Gustavo Isaac Soto Huerta (318201458)
- Brayan Tellez Cruz
- Silverio Martinez Andrés
- Ugalde Santos Atzin

Materia: Bases de Datos
Grupo: 1

Índice

1. Introducción

Como empresa, nuestro propósito es desarrollar proyectos de software que satisfagan las necesidades de nuestros clientes a través de soluciones innovadoras y eficientes. En Universal Harmony, nos dedicamos a desarrollar soluciones de software que integran y armonizan sistemas diversos, mejorando la eficiencia operativa de nuestros clientes. Este documento presenta la documentación del proyecto final, detallando el desarrollo y la implementación del mismo, así como los roles y responsabilidades de cada miembro del equipo.

2. Roles

En esta sección se presentará una breve explicación de los roles asignados a cada miembro del equipo, así como las tareas que cada uno llevará a cabo.

2.1. Database Admin (DBA)

- **Silverio Martinez Andrés**
- **Rivera López David Zaid**

Los administradores de bases de datos son responsables de diseñar, implementar y mantener la base de datos del proyecto, asegurando su eficiencia y seguridad.

2.2. Backend Developer

- **Gustavo Isaac Soto Huerta**
- **Hernandez Ramirez Miguel Angel**

Los desarrolladores backend son responsables de la lógica del servidor, la integración de la base de datos y la creación de API's necesarias para la aplicación.

2.3. Frontend Developer

- **Silverio Martinez Andrés**
- **Hernandez Ramirez Miguel Angel**

- **Brayan Tellez Cruz**

Los desarrolladores frontend se encargan de la implementación de la interfaz de usuario, asegurando una experiencia de usuario amigable y funcional.

2.4. UI Designer

- **Brayan Tellez Cruz**
- **Ugalde Santos Atzin**

Los diseñadores de interfaz de usuario son responsables de la apariencia visual de la aplicación, creando mockups y prototipos para guiar a los desarrolladores frontend.

2.5. Tester

- **Rivera López David Zaid**
- **Ugalde Santos Atzin**

Los testers se encargan de la verificación y validación del sistema, asegurando que cumple con los requisitos especificados y que está libre de errores.

3. Documento de Requerimientos

Se debe permitir el registro de empleados en la base de datos y debe almacenar la información de los empleados incluyendo su RFC, número de empleado, nombre, fecha de nacimiento, teléfonos, edad, domicilio y sueldo. Este registro debe de dividirse en alguno de los 3 cargos posibles: especialidad (para cocineros), horario (para meseros) y rol (para administrativos). Considerar que un empleado puede tener varios puestos y almacenar una foto de los empleados. Además, debe haber un registro de los dependientes de los empleados con su CURP, nombre y parentesco.

Se deben registrar los productos, además de almacenar información de los platillos y bebidas que el restaurante ofrece como descripción, nombre, receta, precio y un indicador de disponibilidad. Debe permitir tener el nombre y descripción de la categoría a la que pertenecen, resaltando que un platillo o bebida solo puede pertenecer a una categoría.

Se debe permitir el registro de órdenes para tener registro del folio de la orden (formato ORD-001), fecha y hora, cantidad total a pagar, registro del

mesero que levantó la orden, cantidad de cada platillo/bebida y precio total a pagar por cada platillo/bebida contenidos en cada orden.

Debe permitir registrar clientes además de almacenar la información de los clientes que soliciten factura, incluyendo su RFC, nombre, domicilio, razón social, email y fecha de nacimiento.

3.1. Actualización Automática

Cada vez que se agregue un producto a la orden deben actualizarse los totales por producto y venta y validar que el producto esté disponible.

3.2. Índices

Crear al menos un índice en la base de datos justificando la elección del tipo de índice y el lugar donde se implementa.

3.3. Consultas Dentro del Sistema

- Mostrar la cantidad de órdenes registradas en el día por un empleado dado (si es mesero) y el total que se ha pagado por dichas órdenes.
- Mostrar todos los detalles del platillo más vendido.
- Obtener el nombre de los productos que no están disponibles.
- Generar una vista automática que contenga la información necesaria para una factura de una orden.
- Dada una fecha o un rango de fechas, regresar el total de ventas y el monto total por las ventas en ese periodo de tiempo.

4. Alcance del Proyecto

4.1. Identificador E1

Documento de Requerimientos
Fecha de entrega: 31/03/2024
Criterios de aceptación:
El documento deberá contener todos los requerimientos que logren satisfacer todas las necesidades de la base de datos del cliente. Los requerimientos deben estar escritos de manera clara y comprensible para todos los miembros del equipo, incluyendo desarrolladores, diseñadores, analistas y clientes.

Los requerimientos deben ser consistentes entre sí y no deben contradecirse. Los requerimientos deben estar priorizados de acuerdo con su importancia y urgencia para el proyecto. Los requerimientos deben poder ser verificados o medidos, permitiendo evaluar si se han cumplido una vez que el sistema esté en funcionamiento. El documento de requerimientos debe ser revisado y aprobado por todos los stakeholders relevantes para asegurar que todos estén de acuerdo con los requerimientos establecidos.

4.2. Identificador E2

Modelo Entidad Relación
Fecha de entrega: 05/04/2024
Criterios de aceptación:
El modelo entidad-relación debe representar con precisión los requerimientos del sistema, incluyendo todas las entidades, atributos y relaciones necesarias. El modelo debe ser claro y fácil de entender para todos los miembros del equipo, incluidos desarrolladores, diseñadores, analistas y clientes. El modelo debe ser coherente, es decir, no debe contener contradicciones entre entidades, atributos o relaciones. El modelo debe incluir mecanismos para mantener la integridad de los datos, como claves primarias, claves foráneas y restricciones de integridad que satisfagan las consultas necesarias claras en los requerimientos. El modelo debe seguir los estándares de diseño de bases de datos y las mejores prácticas de la industria.

4.3. Identificador E3

Modelo Relacional
Fecha de entrega: 10/04/2024
Criterios de aceptación:
El modelo relacional debe representar con precisión los requerimientos del sistema, incluyendo todas las tablas, columnas, claves primarias y foráneas, y restricciones necesarias. Cada tabla debe tener una clave primaria definida que identifique de manera única a cada fila de datos. Las relaciones entre tablas deben estar definidas con claves foráneas, asegurando la integridad referencial. El modelo debe incluir índices apropiados para mejorar el rendimiento de las consultas y operaciones en la base de datos. El modelo debe estar optimizado para el rendimiento y la eficiencia, evitando problemas como la duplicación de datos y las operaciones costosas. El modelo debe seguir los estándares de diseño de bases de datos relacionales y las mejores prácticas de la industria. El modelo debe incluir relaciones apropiadas para mantener la

integridad de los datos, como restricciones de unicidad, integridad referencial y validaciones de datos.

4.4. Identificador E4

Propuestas de vistas del sistema
Fecha de entrega: 20/04/2024
Criterios de aceptación:
Las vistas deben proporcionar información relevante para los usuarios o procesos que las consultarán. Deben ayudar a resolver problemas específicos o responder a preguntas concretas. Las vistas deben estar optimizadas para un rendimiento adecuado. Las consultas subyacentes deben ser eficientes y evitar operaciones costosas como escaneos completos de tablas o uniones complicadas. Las vistas deben cumplir con las políticas de seguridad y privacidad del sistema. Los usuarios solo deben tener acceso a los datos para los que están autorizados. Se debe considerar el control de acceso para asegurarse de que solo los usuarios autorizados puedan acceder a las vistas.

4.5. Identificador E5

Creación de la Base de Datos
Fecha de entrega: 30/04/2024
Criterios de aceptación:
La base de datos debe satisfacer los requisitos funcionales y no funcionales del usuario o cliente. Debe cumplir con las especificaciones de diseño previamente acordadas. La estructura de la base de datos debe ser coherente con el modelo de datos diseñado. Las tablas, columnas y relaciones deben estar bien definidas y alineadas con los requisitos de la aplicación. La base de datos debe ser capaz de manejar la carga esperada de datos y consultas de manera eficiente. Debe ser posible escalar la base de datos según sea necesario a medida que crezca la cantidad de datos o la carga de trabajo. La base de datos debe ser fácil de usar para los desarrolladores y usuarios finales. Las consultas y operaciones deben ser intuitivas y eficientes.

4.6. Identificador E6

Programación a nivel base de datos
Fecha de entrega: 31/04/2024
Criterios de aceptación:
Asegurar que todos los datos insertados, actualizados o eliminados respeten las restricciones de integridad definidas en la base de datos. Validar que las

llaves primarias y foráneas se mantengan consistentes. Verificar que cualquier producto agregado a una orden esté marcado como disponible. Levantar una excepción y abortar la operación si el producto no está disponible. Actualizar automáticamente los totales por producto (detallePrc) al agregar o modificar un producto en la orden. Permitir consultas que tomen una fecha o un rango de fechas (fecha de inicio y fecha de fin) para retornar el total de ventas y el monto total de ventas en el período especificado.

4.7. Identificador E7

Pruebas Unitarias
Fecha de entrega: 07/05/2024
Criterios de aceptación:
Llaves Primarias y Foráneas: La base de datos impide la inserción de registros duplicados para llaves primarias y asegura que las llaves foráneas referencian correctamente a registros existentes. Se validan restricciones de check que aseguran que los valores de las columnas cumplen con las condiciones especificadas. Los índices se crean correctamente en las columnas especificadas. Las consultas y vistas manejan adecuadamente las condiciones de error como datos faltantes o entradas no válidas. La función/procedimiento devuelve los resultados esperados para una variedad de entradas válidas.

4.8. Identificador E8

Documentación
Fecha de entrega: 15/05/2024
Criterios de aceptación:
Configuración Inicial: Proporcionar instrucciones claras sobre cómo configurar y desplegar la base de datos, incluyendo requisitos previos y pasos de instalación. Procedimientos de Mantenimiento: Incluir procedimientos recomendados para el mantenimiento regular de la base de datos, como respaldo y recuperación, optimización y actualizaciones. Ejemplos: Incluir ejemplos de consultas, inserciones, actualizaciones y eliminaciones para ilustrar cómo interactuar con la base de datos. Casos de Uso: Documentar casos de uso comunes y mejores prácticas para la interacción con la base de datos. Formato Consistente: Utilizar un formato consistente para documentar tablas, columnas, funciones, procedimientos, triggers e índices. Organización Lógica: La documentación debe estar organizada de manera lógica, facilitando la navegación y búsqueda de información específica.

5. Sección Previa

En esta sección se documenta la información de las reuniones y las actividades realizadas por el project manager.

5.1. Primera sesión (14-04-2024)

En nuestra primera sesión, se llevó a cabo la asignación correspondiente de tareas para cada uno de los integrantes del equipo, y que se muestra en este documento, al inicio del mismo. Además, se realizó un pequeño bosquejo a la realización de nuestro "Modelo Entidad Relación", donde se tuvieron en cuenta los siguientes puntos importantes para la realización de este mismo:

- La entidad empleado tendrá una generalización a los puestos de trabajo que este puede tener. Así como esta generalización tiene una relación total y de traslape (de acuerdo a los requerimientos) para cada uno de los subtipos presentes.
- Los nombres de los empleados, clientes y dependientes estarán lo más detallado posible (haciendo referencia tanto a su nombre de pila, apellido paterno y apellido materno, el cual será un atributo opcional).
- El empleado puede o no necesariamente tener un dependiente.
- Al igual que el nombre, el domicilio de los clientes y de los empleados estará lo mayor desglosado posible (con calle, código postal, colonia, número exterior y número interior siendo este un atributo opcional).
- La relación entre orden y producto tendrá como atributos cantidad y precio, para un mejor manejo de ambos al futuro.
- La entidad producto tendrá el atributo numVentas para, a futuro, consultar qué producto ha sido el más vendido de nuestro registro.

Por último, se dió finalizada la sesión presentando los pequeños bosquejos realizados durante la sesión.

5.2. Segunda sesión (17-04-2024)

De acuerdo a los bosquejos realizados en la sesión pasada, se realizó el Modelo Entidad Relación definitivo, el cual, después de varias modificaciones, quedó de la forma en la que se presenta en el diseño.

5.3. Tercera Sesión (21-04-2024)

Se empezó a plantear a grosso modo el Modelo Relacional (MR), a partir de nuestro Modelo Entidad Relación (MER). Principalmente, se acordaron los tipos de datos para cada uno de los atributos y el correcto mapeo del MER a nuestro MR.

5.4. Cuarta Sesión

Después de haber abordado correctamente los errores puntuales del MR realizado, y haberlos corregido, se empezó a implementar el modelo físico junto a todos sus requerimientos solicitados por el profesor.

5.5. Quinta Sesión

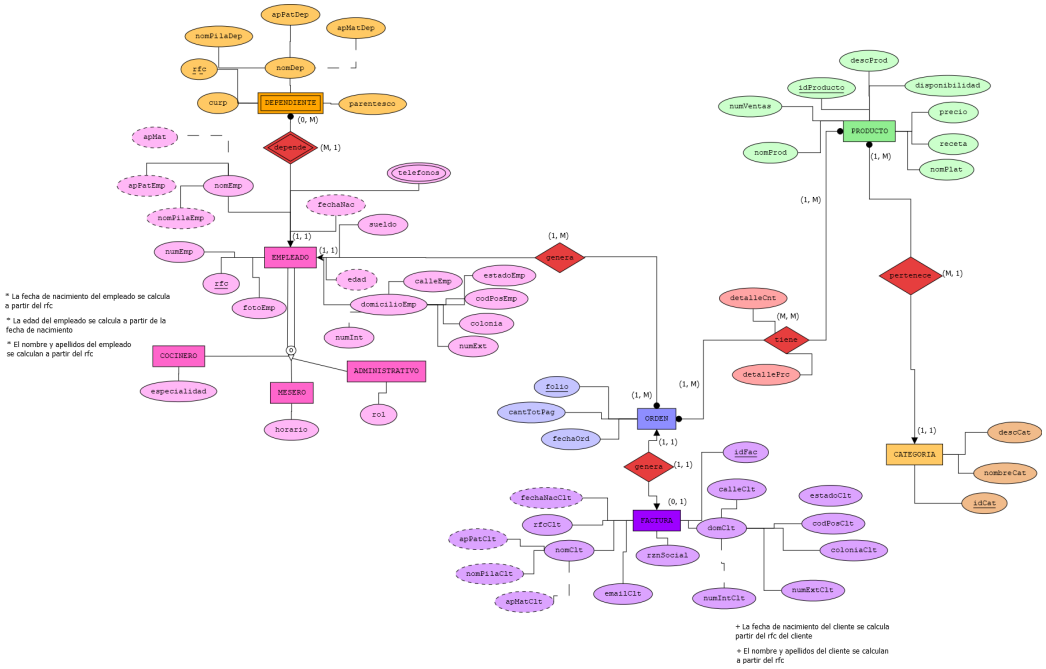
Se presentó la versión final del modelo físico ya implementado junto con todos los requerimientos que solicitó el profesor, el cual, después de la verificación y aprobación de todo el equipo, así como su debido testeo, no se le realizó ningún cambio después de eso.

6. Primera Sección: Database

En esta sección se explica el desarrollo del producto desde la perspectiva de la base de datos, incluyendo diagramas y explicaciones detalladas.

6.1. Desarrollo del Modelo Entidad-Relación

Aquí se presenta el diseño del Modelo Entidad-Relación:



6.2. Implementación del Modelo Relacional

Explicación de cómo se transformó el Modelo Entidad-Relación en un Modelo Relacional, incluyendo las tablas creadas, sus atributos y las relaciones entre ellas.

6.3. Normalización de Tablas

Descripción de las tablas normalizadas:

- **Tabla ÇATEGORÍA":** Relación: Cada categoría (ïdCat") tiene un nombre ("nombreCat") y una descripción ("desCat"). Cumplimiento de 3FN: Todos los atributos dependen directamente de la clave primaria ïdCat", sin dependencias transitivas ni parciales.

- **Tabla "TELEFONO":** Relación: Cada empleado (rfcTel") tiene asociado un número de teléfono ("telefono"). Cumplimiento de 3FN: El número de teléfono depende directamente de la clave primaria rfcTel", sin dependencias transitivas ni parciales.

- **Tabla .ᴱᴹPLEADO":** Relación: Cada empleado (rfc") tiene un número de empleado ("numEmp"), una foto ("fotoEmp"), nombre ("nomPilaEmp", .ªpPatEmp", .ªpMatEmp"), fecha de nacimiento ("fecNa-

cEmp"), sueldo ("sueldo"), etc. Cumplimiento de 3FN: Todos los atributos dependen directamente de la clave primaria rfc", sin dependencias transitivas ni parciales.

- **Tabla "FACTURA":** Relación: Cada factura (ïdFac") está asociada a un dependiente (rfcDep"), que tiene un correo electrónico (.emailClt"), razón social (rznSocial"), etc. Cumplimiento de 3FN: Todos los atributos dependen directamente de la clave primaria ïdFac", sin dependencias transitivas ni parciales.

- **Tablas .ADMINISTRATIVO", "MESEROz ÇOCINERO":** Relación: Cada empleado en estas tablas tiene un rol específico (rol.en .ADMINISTRATIVO", "horario.en "MESERO", .especialidad.en ÇOCINERO"). Cumplimiento de 3FN: Todos los atributos dependen directamente de la clave primaria correspondiente, sin dependencias transitivas ni parciales.

- **Tabla "DEPENDIENTE":** Relación: Cada dependiente (rfc") tiene una CURP (çurp"), un nombre ("nomPilaDep", .apPatDep", .apMatDep"), y un parentesco ("parentesCo"). Cumplimiento de 3FN: Todos los atributos dependen directamente de la clave primaria rfc", sin dependencias transitivas ni parciales.

- **Tabla ÇOBORD":** Relación: Cada registro en esta tabla relaciona un empleado (rfc") con una orden ("folio"). Cumplimiento de 3FN: Todos los atributos dependen directamente de las claves primarias rfcz "folio", sin dependencias transitivas ni parciales.

- **Tabla .ORDPROD":** Relación: Cada registro en esta tabla relaciona una orden ("folio") con un producto (ïdProducto"), y tiene una cantidad y precio de detalle. Cumplimiento de 3FN: Todos los atributos dependen directamente de las claves primarias "folio.eidProducto", sin dependencias transitivas ni parciales.

- **Tabla "PRODUCTO":** Relación: Cada producto (ïdProducto") tiene un número de ventas ("numVentas"), descripción ("descProd"), precio ("precio"), etc. Cumplimiento de 3FN: Todos los atributos dependen directamente de la clave primaria ïdProducto", sin dependencias transitivas ni parciales.

- **Tabla .ORDEN":** Relación: Cada orden ("folio") tiene una cantidad total pagada (çantTotPag"), una fecha ("fechaOrd"), etc., y está asociada a un mesero (rfcMsr"). Cumplimiento de 3FN: Todos los atributos

dependen directamente de la clave primaria "folio", sin dependencias transitivas ni parciales. Además, la referencia al mesero se hace a través de la clave externa rfcMsr", evitando así la dependencia transitiva.

6.4. Modelo Relacional

A partir del Modelo Entidad - Relación y a la normalización realizada en los puntos anteriores, se hizo la transformación al Modelo Relacional, el cual quedó de la siguiente manera para cada una de las relaciones:

- **PRODUCTO**

```
PRODUCTO{
    idProducto  integer  PK,
    numVentas  integer,
    descProd  text,
    disponibilidad  boolean,
    precio  numeric(10, 2),
    nomProd varchar(100),
    receta  text,
    idCat  smallint  FK,
}
```

- **CATEGORÍA**

```
CATEGORIA{
    idCat  smallint  PK,
    nombreCat  varchar(100),
    descCat  text,
}
```

- **ORDEN**

```
ORDEN{
    folio  varchar(20)  PK,
    cantTotPag  numeric(10, 2),
    fechaOrd  timestamp,
    rfcMsr  varchar(13)  FK,
    idFac integer
}
```

- **FACTURA**

```
FACTURA{
    idFac serial PK,
    rfcClt  char(13),
    nomPilaClt  varchar(100) C,
    apPatClt  varchar(100) C,
    apMatClt varchar(100) C  NULL,
    fechaNacClt  date C,
    emailClt  varchar(225),
    rznSocial  text,
    calleClt  varchar(200),
    codPosClt  char(5),
    colClt  varchar(100),
    numExtClt  smallint,
    numIntClt  smallint  NULL,

}
```

- **ORDPROD**

```
ORDPROD{
    [folio  varchar(20),
    idProducto  smallint] FK PK,
    detalleCnt  smallint,
    detallePrc  numeric(10, 2)
}
```

- **EMPLEADO**

```
EMPLEADO{
    rfc  char(13)  PK,
    nomPilaEmp  varchar(100) C,
    apPatEmp  varchar(100) C,
    apMatEmp  varchar(100) C,
    fotoEmp  bytea,
    numEmp  integer,
    edad  numeric(2)  C,
    fechaNacEmp  date  C,
    sueldo  numeric(8, 2),
    estadoEmp varchar(100),
    calleEmp  varchar(200),
    codPosEmp  char(5),
    colEmp  varchar(100),
    numExt  smallint,
    numInt  smallint  NULL,
}
```

- **COCINERO**

```
COCINERO{
    rfcCoc  char(13)  FK  PK,
    especialidad  text,
}
```

- **MESERO**

```
MESERO{
    rfcMsr  char(13)  FK  PK,
    horario  timestamp,
}
```

- **ADMINISTRATIVO**

```
ADMINISTRATIVO{
    rfcAdm  char(13)  FK  PK,
    rol  varchar(100),
}
```

- **TELÉFONO**

```
TELEFONO{
    rfcTel  varchar(13)  FK  PK,
    telefono  bigint,

}
```

- **DEPENDIENTE**

```
DEPENDIENTE{
    [rfc  varchar(13)  FK,
    curp  varchar(18)  D] PK
    nomPilaDep  varchar(100),
    apPatDep  varchar(100),
    apMatDep  varchar(100)  NULL,
    parentesco  varchar(100)
}
```

- **COBORD**

```
COBORD{
    [rfc char(13),
    folio varchar(20)] FK PK
}
```

6.5. Modelo Físico

Ahora, ya con nuestras tablas del Modelo Relacional bien construidas, se procedió a la construcción del Modelo Físico, el cual, después, se implementará a PostgreSQL. Entonces, nuestras tablas en DDL, quedaron de la siguiente manera (tomando en cuenta los tipos de dato, llaves primarias y foráneas y ciertas restricciones):

- Para CATEGORÍA
- Para TELÉFONO
- Para EMPLEADO
- Para FACTURA
- Para ADMINISTRATIVO

- Para MESERO
- Para COCINERO
- Para DEPENDIENTE
- Para COBORD
- Para ORDPROD
- Para PRODUCTO
- Para ORDEN

En este caso, para no tener problemas al implementar las llaves foráneas y primarias en las relaciones implementadas anteriormente, se agregaron los "CONSTRAINT" necesarios para cada una de las relaciones, quedando de la siguiente manera:

```
ALTER TABLE "ADMINISTRATIVO"
ADD CONSTRAINT "ADMINISTRATIVO_rfcAdm_fkey" FOREIGN KEY ("rfcAdm") REFERENCES "EMPLEADO" (rfc);

ALTER TABLE "PRODUCTO"
ADD CONSTRAINT "PRODUCTO_idCat_fkey" FOREIGN KEY ("idCat") REFERENCES "CATEGORIA" ("idCat");

ALTER TABLE "COBORD"
ADD CONSTRAINT "COBORD_folio_fkey" FOREIGN KEY (folio) REFERENCES "ORDEN" (folio);

ALTER TABLE "COBORD"
ADD CONSTRAINT "COBORD_rfc_fkey" FOREIGN KEY (rfc) REFERENCES "EMPLEADO" (rfc);

ALTER TABLE "COCINERO"
ADD CONSTRAINT "COCINERO_rfcCoc_fkey" FOREIGN KEY ("rfcCoc") REFERENCES "EMPLEADO" (rfc);

ALTER TABLE "DEPENDIENTE"
ADD CONSTRAINT "DEPENDIENTE_rfc_fkey" FOREIGN KEY (rfc) REFERENCES "EMPLEADO" (rfc);

ALTER TABLE "FACTURA"
ADD CONSTRAINT "FACTURA_idFac_fkey" FOREIGN KEY ("idFac") REFERENCES "ORDEN" ("idFac");

ALTER TABLE "MESERO"
ADD CONSTRAINT "MESERO_rfcMsr_fkey" FOREIGN KEY ("rfcMsr") REFERENCES "EMPLEADO" (rfc);

ALTER TABLE "ORDEN"
ADD CONSTRAINT "ORDEN_rfcMsr_fkey" FOREIGN KEY ("rfcMsr") REFERENCES "EMPLEADO" (rfc);

ALTER TABLE "ORDPROD"
ADD CONSTRAINT "ORDPROD_folio_fkey" FOREIGN KEY (folio) REFERENCES "ORDEN" (folio);

ALTER TABLE "ORDPROD"
ADD CONSTRAINT "ORDPROD_idProducto_fkey" FOREIGN KEY ("idProducto") REFERENCES "PRODUCTO" ("idProducto");

ALTER TABLE "TELEFONO"
ADD CONSTRAINT "TELEFONO_rfcTel_fkey" FOREIGN KEY ("rfcTel") REFERENCES "EMPLEADO" (rfc);
```

6.6. Funciones, Triggers y Vistas

Después, se crearon las funciones, triggers y vistas necesarias para cumplir con los requerimientos específicos que solicite nuestro cliente, resolviéndose de la siguiente manera:

6.6.1. Actualización de Totales y Validación de Disponibilidad

Como primer punto, como modelo de negocio se nos solicitó que cada que se agregue un producto a la orden, debe actualizarse los totales (por producto y venta), así como validar que el producto esté disponible. Este punto se atendió de la siguiente manera:

```
-- Calculando el total de prod en ORDPROD
CREATE OR REPLACE FUNCTION calculate_order_product_total() RETURNS TRIGGER AS $$
DECLARE
    product_price NUMERIC(10, 2);
BEGIN
    SELECT precio INTO product_price FROM "PRODUCTO" WHERE "idProducto" = NEW."idProducto";
    NEW."detallePrc" := NEW."detalleCnt" * product_price;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER update_order_product_total
BEFORE INSERT OR UPDATE ON "ORDPROD"
FOR EACH ROW
EXECUTE FUNCTION calculate_order_product_total();
```

```
CREATE OR REPLACE FUNCTION check_product_availability() RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM "PRODUCTO"
        WHERE "idProducto" = NEW."idProducto" AND disponibilidad = TRUE
    ) THEN
        RAISE EXCEPTION 'El producto no está disponible.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_availability_before_insert_or_update
BEFORE INSERT OR UPDATE ON "ORDPROD"
FOR EACH ROW
EXECUTE FUNCTION check_product_availability();
```

6.6.2. Optimización del Acceso a Productos Disponibles

Como segunda solicitud de nuestro cliente se nos pide que la forma en la que se accede a los productos disponibles sea más eficiente, para evitar errores al momento de generar una orden con algún producto que no se encuentra disponible. Para atender este punto se creó un índice para enlistar a los productos disponibles en la base de datos:

```
CREATE INDEX ixDispoProd ON "PRODUCTO" (disponibilidad) WHERE disponibilidad = TRUE;
```

6.6.3. Consulta de Órdenes Registradas por Empleado

Como tercera cláusula de nuestro cliente, nos solicita que dado un número de empleado, se muestre la cantidad de órdenes que ha registrado en el día así como el total que se ha pagado por dichas órdenes. Si no se trata de un mesero, mostrar un mensaje de error. Este punto se atendió de la siguiente manera:

```
CREATE OR REPLACE FUNCTION empMesero(numEmp INTEGER, OUT ordenesAtendidas INTEGER, OUT totalPagado NUMERIC)
RETURNS RECORD AS
$$
DECLARE
    esMesero BOOLEAN;
BEGIN
    -- Verificar si el empleado es un mesero
    SELECT EXISTS(SELECT 1 FROM "MESERO" WHERE "rfcMsr" = (SELECT rfc FROM "EMPLEADO" WHERE "numEmp" = numEmp)) INTO

    IF esMesero THEN
        -- Contar la cantidad de órdenes registradas por el mesero en el día
        SELECT COUNT(*), COALESCE(SUM("cantTotPag"), 0)
        INTO ordenesAtendidas, totalPagado
        FROM "ORDEN"
        WHERE "rfcMsr" = (SELECT rfc FROM "EMPLEADO" WHERE "numEmp" = numEmp);

    ELSE
        -- Mostrar un mensaje de error si no es un mesero
        RAISE EXCEPTION 'El numero de empleado introducido no corresponde a un mesero';
    END IF;
END;
$$
LANGUAGE plpgsql;
```

6.6.4. Vista del Platillo Más Vendido

Basado en el modelo de negocio de nuestro cliente, se nos solicita generar una vista que muestre todos los detalles del platillo más vendido. Este punto se abordó de la siguiente manera:

```sql
CREATE VIEW vistaProdMasVendido AS
SELECT * FROM "PRODUCTO"
WHERE "numVentas" = (SELECT MAX("numVentas") FROM "PRODUCTO");
```

6.6.5. Consulta de Productos No Disponibles

A nuestro cliente también le interesa permitir obtener el nombre de aquellos productos que no estén disponibles. Este punto se abordó de la siguiente manera:

```sql
-- Devolviendo los productos que no se encuentran disponibles
CREATE OR REPLACE FUNCTION prodNoDisp()
RETURNS TABLE (
    productoNoDisponible varchar(100)
) AS
$$
BEGIN
    RETURN QUERY
    SELECT "nomProd" FROM "PRODUCTO"
    WHERE disponibilidad = FALSE;
END;
$$
LANGUAGE plpgsql;

-- Usarla como
SELECT * FROM prodNoDisp();
```

6.6.6. Vista de Detalles de Factura

El cliente muestra interés en la producción de una vista que se asemeje lo suficiente y contenga los datos de una factura. Este punto se atendió extrayendo los datos de la factura a una factura de las diferentes tablas gracias a sus conexiones:

```sql
CREATE OR REPLACE VIEW vistaFactura AS
SELECT
    o.folio,
    o."cantTotPag",
    o."fechaOrd",
    op."idProducto",
    p."nomProd",
    op."detalleCnt",
    op."detallePrc",
    f."idFac",
    f."rfcClt",
    f."nomPilaClt",
    f."apPatClt",
    f."apMatClt",
    f."fechaNacClt",
    f."emailClt",
    f."rznSocial",
    f."calleClt",
    f."codPosClt",
    f."colClt",
    f."numExtClt",
    f."numIntClt"
FROM
    "ORDEN" o
JOIN
    "ORDPROD" op ON o.folio = op.folio
JOIN
    "PRODUCTO" p ON p."idProducto" = op."idProducto"
RIGHT JOIN
    "FACTURA" f ON f."idFac" = o."idFac";
```

6.6.7. Consulta de Ventas por Período de Tiempo

Como última solicitud de nuestro cliente, nos especifica generar una consulta que, dada una fecha, o una fecha de inicio y fecha de fin, regrese el total del número de ventas y el monto total por las ventas en ese periodo de tiempo. Nota: Con el producto se hace referencia a los alimentos y bebidas. Este punto se atendió de la siguiente manera:

```sql
CREATE OR REPLACE FUNCTION ventas(fechaI DATE, fechaF DATE)
RETURNS TABLE(totVentas INT, montTot NUMERIC) AS $$
BEGIN
    RETURN QUERY
    SELECT
        CAST(COUNT(DISTINCT folio) AS INTEGER) AS totVentas,
        COALESCE(SUM("cantTotPag"), 0) AS montTot
    FROM
        "ORDEN"
    WHERE
        "fechaOrd" BETWEEN fechaI AND fechaF;
END;
$$ LANGUAGE plpgsql;
```

6.7. Restricciones Adicionales

Gracias a la comunicación continua con nuestro cliente, se nos notificaron restricciones a nuestro modelo que cumpliremos una por una.

6.7.1. Formato del Folio de Orden

El folio de la orden debe tener un formato similar a ORD-001, prefijo ORD, seguido de un guión y un número secuencial. Este punto se solucionó de la siguiente manera:

```sql
177 ALTER TABLE "ORDEN"
178 ALTER COLUMN folio SET DEFAULT 'ORD-' || LPAD(nextval('ordenFolioSec')::TEXT, 4, '0');
```

6.7.2. Estructura del Atributo Domicilio

Donde esté presente el atributo domicilio, está compuesto por estado, código postal, colonia, calle y número. Esto lo podemos visualizar en la creación de nuestras tablas:

```
"calleEmp" varchar(200) NOT NULL,
"codPosEmp" char(5) NOT NULL,
"colEmp" varchar(100) NOT NULL,
"numExt" smallint NOT NULL,
"numInt" smallint,
```

6.7.3. Estructura del Atributo Nombre

Donde esté presente el atributo nombre, está compuesto por nombre, apellido paterno y materno. Esta implementación la podemos ver en la creación de las tablas ya que los atributos demuestran ser compuestos por otros:

```
CREATE TABLE "EMPLEADO"(
  rfc char(13) NOT NULL,
  "numEmp" integer NOT NULL,
  "fotoEmp" bytea NOT NULL,
  "nomPilaEmp" varchar(100) NOT NULL,
  "apPatEmp" varchar(100) NOT NULL,
  "apMatEmp" varchar(100),
```


```
CREATE TABLE "EMPLEADO"(
  rfc char(13) NOT NULL,
  "numEmp" integer NOT NULL,
  "fotoEmp" bytea NOT NULL,
  "nomPilaEmp" varchar(100) NOT NULL,
  "apPatEmp" varchar(100) NOT NULL,
  "apMatEmp" varchar(100),
```


Hasta aquí podemos verificar que tanto las condiciones de negocio como las restricciones de almacenamiento de datos fueron estratégicamente implementadas en la base de datos desarrollada por nuestro equipo.

7. Segunda Sección: Full Stack Developer

Esta sección describe cómo se implementaría la integración entre la base de datos y una aplicación imaginaria, detallando los métodos y tecnologías utilizadas.

7.1. Integración de la Base de Datos

Descripción de cómo se integraría la base de datos con la aplicación, incluyendo ejemplos de consultas SQL y su implementación en el código

backend:

```
"calleEmp" varchar(200) NOT NULL,
"codPosEmp" char(5) NOT NULL,
"colEmp" varchar(100) NOT NULL,
"numExt" smallint NOT NULL,
"numInt" smallint,
```

7.2. Desarrollo de API

Explicación de cómo se desarrollaría una API para interactuar con la base de datos y la aplicación frontend:

```
CREATE TABLE "EMPLEADO"(
  rfc char(13) NOT NULL,
  "numEmp" integer NOT NULL,
  "fotoEmp" bytea NOT NULL,
  "nomPilaEmp" varchar(100) NOT NULL,
  "apPatEmp" varchar(100) NOT NULL,
  "apMatEmp" varchar(100),
```


```
CREATE TABLE "EMPLEADO"(
  rfc char(13) NOT NULL,
  "numEmp" integer NOT NULL,
  "fotoEmp" bytea NOT NULL,
  "nomPilaEmp" varchar(100) NOT NULL,
  "apPatEmp" varchar(100) NOT NULL,
  "apMatEmp" varchar(100),
```


8. UI Designer

En esta sección se presenta una explicación detallada sobre el diseño de la interfaz de usuario sin incluir los mockups.

8.1. Diseño de la Interfaz de Usuario

Descripción detallada de las decisiones de diseño, colores, tipografía y la disposición de los elementos en la interfaz:

9. Cuarta Sección: Testeos

Esta sección introduce los procesos de prueba realizados por Miguel y Atzin, detallando los tipos de pruebas, casos de prueba y los resultados ob-

tenidos.

9.1. Pruebas Unitarias

Descripción de las pruebas unitarias realizadas, incluyendo ejemplos de casos de prueba y sus resultados:

9.2. Pruebas de Integración

Descripción de las pruebas de integración realizadas, destacando cómo se aseguraron de que los diferentes componentes del sistema funcionaran correctamente juntos:

10. Conclusión

Este proyecto ha sido un éxito gracias a la dedicación y esfuerzo de todo el equipo. Hemos logrado desarrollar una solución de software integral que satisface plenamente los requisitos del cliente. Los objetivos específicos, como la implementación de una base de datos eficiente, la creación de API's funcionales, y el diseño de una interfaz de usuario intuitiva, fueron alcanzados con éxito. Además, las pruebas realizadas garantizan la fiabilidad y el rendimiento del sistema, asegurando que todos los componentes funcionan de manera cohesiva y eficiente.

10.1. Resumen del Proyecto

En este proyecto, se logró diseñar e implementar una base de datos robusta y eficiente que cumple con todos los requisitos especificados por el cliente. Los roles y responsabilidades del equipo fueron claramente definidos, lo que permitió una ejecución organizada y exitosa de cada una de las tareas. Se desarrollaron y normalizaron tablas, se implementaron funciones, triggers y vistas necesarias, y se crearon índices que optimizan el rendimiento de las consultas. Además, se diseñó una interfaz de usuario amigable y funcional, y se realizaron pruebas exhaustivas para asegurar la calidad del producto final.

10.2. Compromiso de la Empresa

En Universal Harmony, nos comprometemos a ofrecer soluciones de software de alta calidad que satisfagan las necesidades de nuestros clientes. Este

proyecto refleja nuestro enfoque en la innovación y la eficiencia, asegurando que nuestros productos no solo cumplan con los estándares más altos de la industria, sino que también superen las expectativas de nuestros clientes. Estamos dedicados a mantener una comunicación abierta y continua con nuestros clientes, garantizando que cada proyecto sea una colaboración exitosa y satisfactoria. Nos enorgullece nuestro trabajo y seguimos comprometidos a proporcionar un servicio excepcional y resultados de calidad.