

Proyecto Final Bases de Datos

Beltran Garcia Fernando Ivan Gómez Enríquez Agustín Madera Fuente Jose Angel Walls Chávez Luis

27 de noviembre de 2023

Índice

1. Introducción	1
2. Plan de Trabajo	2
2.1. Organigrama	2
3. Diseño	4
4. Implementación	7
5. Presentación	14
6. Conclusiones	15

1. Introducción

La digitalización de los negocios es un proceso cada vez más necesario para las empresas de todos los tamaños. En el sector de la restauración, la implementación de un sistema informático que permita automatizar los procesos y facilitar la gestión de la información puede suponer un importante ahorro de tiempo y recursos.

En este contexto, el diseño de una base de datos adecuada es un elemento clave para el éxito de cualquier proyecto de digitalización. La base de datos debe ser capaz de almacenar toda la información relevante para el negocio, de forma que sea accesible y fácil de gestionar.

El texto que se presenta en este documento describe los requerimientos para el diseño de una base de datos para un restaurante. Los requerimientos incluyen la información de los empleados, los platillos y bebidas que ofrece el restaurante, las órdenes de los clientes y la información de los clientes.

La base de datos deberá cumplir con los siguientes principios de diseño:

- Atributos atómicos: Los atributos deben representar conceptos simples y discretos.

- Independencia de datos: Los datos deben estar organizados de forma que sean independientes de los programas que los utilizan.
- Normalización: La base de datos debe estar normalizada para evitar la redundancia de datos.
- Unicidad: Los datos deben ser únicos para cada entidad.

La base de datos deberá ser implementada en PostgreSQL, un sistema de gestión de bases de datos relacionales. Una vez diseñada y lista la base de datos, se deberá elegir la implementación de uno de los siguientes puntos:

- Consultar la información general de los empleados, incluyendo su fotografía por medio de una app móvil o web.
- Generar un dashboard que permita visualizar, al menos: Ingresos del mes, platillos sin existencia, horas pico en que se registran más órdenes.
- Ingresar, a partir de archivos de texto, la información registrada durante el día a otra base de datos. Se debe orquestrar la inserción de cada archivo y tener validaciones del flujo en caso de errores.

La implementación de cualquiera de estos puntos permitirá ampliar las funcionalidades de la base de datos y mejorar la gestión del restaurante.

2. Plan de Trabajo

El equipo trabajó utilizando las siguientes herramientas:

1. WhatsApp. Para organizar el proyecto y consultas rápidas.
2. Google DOCS. Para trabajar al mismo tiempo la documentación del proyecto.
3. LATEX. Una vez finalizado el documento se pasó a Latex donde se le dio el formato adecuado.
4. GitHub. Para compartir códigos fuente de los modelos, scripts de la base de datos, códigos y enlaces para la página web.
5. Power Point. Para realizar la presentación del proyecto.

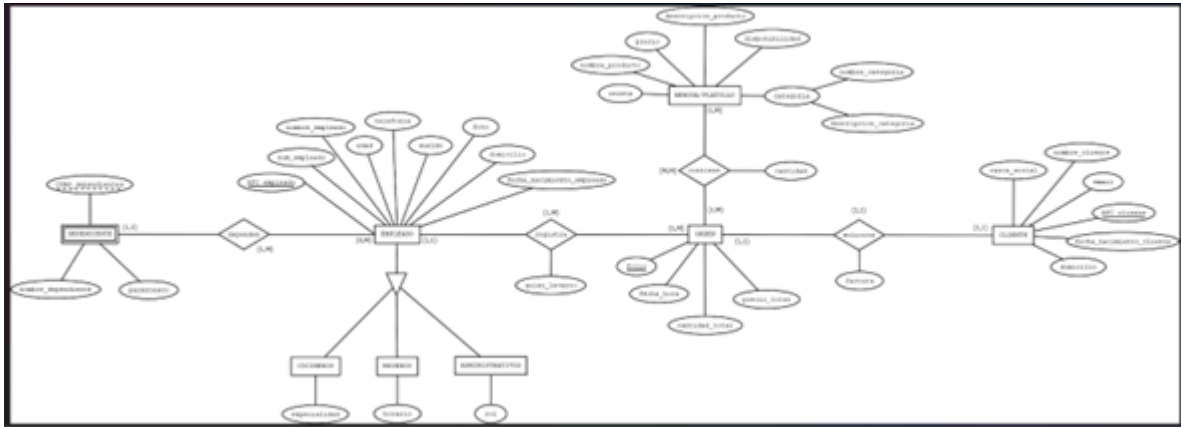
2.1. Organigrama

Se presenta un organigrama de las actividades:

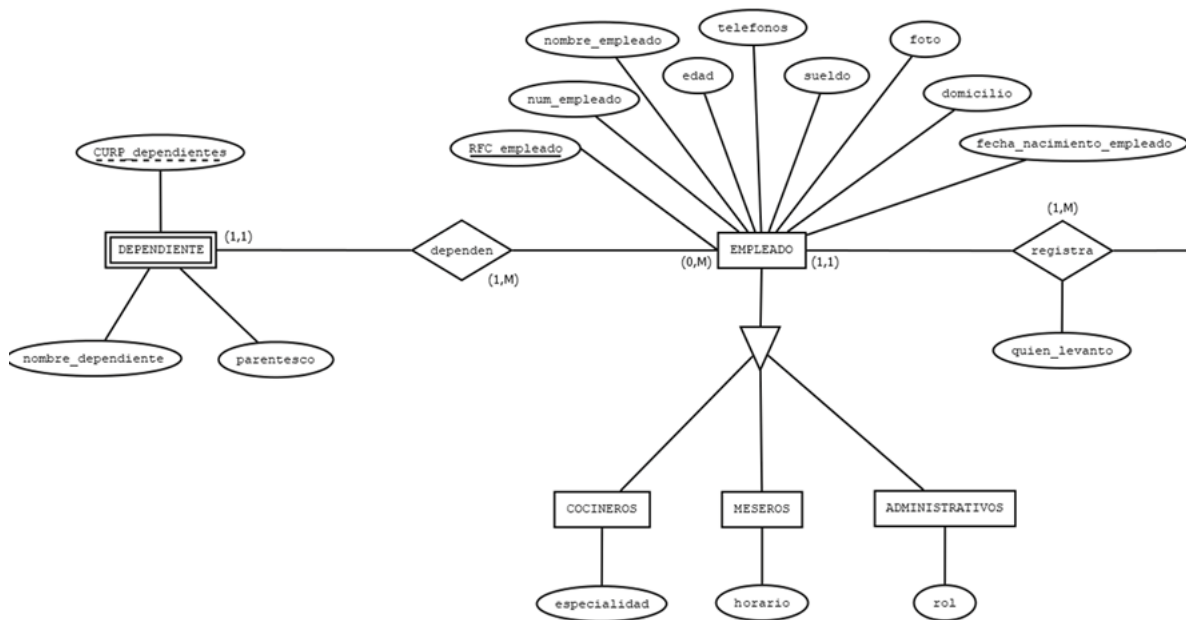
NOMBRE	ACTIVIDAD	DESCRIPCIÓN
Beltran Garcia Fernando Ivan.	<ul style="list-style-type: none"> ■ MODELO RELACIONAL ■ SCRIPT PROGRAMACIÓN EN SQL (POSTGRESQL) ■ DOCUMENTACIÓN ■ PRESENTACIÓN 	Para la realización de la documentación se utilizó un editor de LaTeX en línea llamado Overleaf
Gómez Enríquez Agustín.	<ul style="list-style-type: none"> ■ MER ■ MODELO RELACIONAL ■ SCRIPT CREACION Y PROGRAMACIÓN EN SQL (POSTGRESQL) ■ DOCUMENTACIÓN ■ PRESENTACIÓN 	<p>El alumno realizó el modelo entidad relación en el programa DIA.</p> <p>Para el modelo relacional se utilizó PGMODELER.</p> <p>Para la programación del código en SQL, se trabajó desde PGADMIN4 en el servidor asignado por el profesor y también de forma local para pruebas de errores.</p>
Madera Fuente Jose Angel	<ul style="list-style-type: none"> ■ SCRIPT AGREGADO DE INFORMACIÓN ■ DOCUMENTACIÓN ■ PRESENTACIÓN 	
Walls Chávez Luis Fernando.	<ul style="list-style-type: none"> ■ IMPLEMENTACIÓN APP WEB ■ SCRIPT AGREGADO DE INFORMACIÓN ■ DOCUMENTACIÓN ■ PRESENTACIÓN 	

3. Diseño

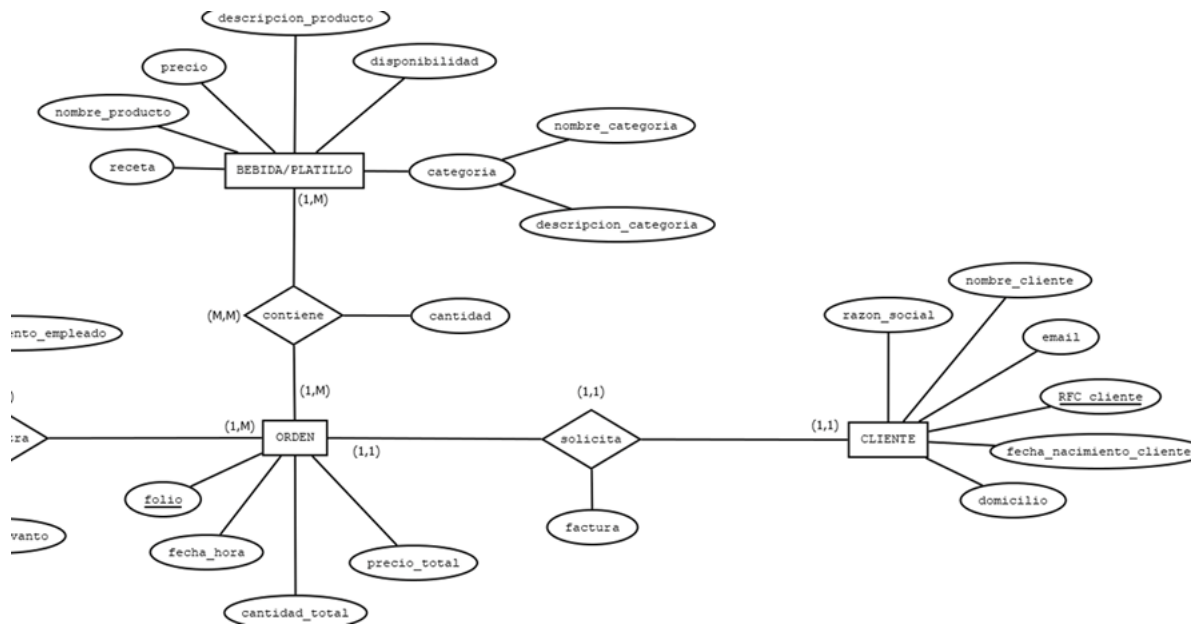
Considerando los requerimientos del proyecto se realizó el siguiente modelado del modelo entidad relación. Se creó en DIA:



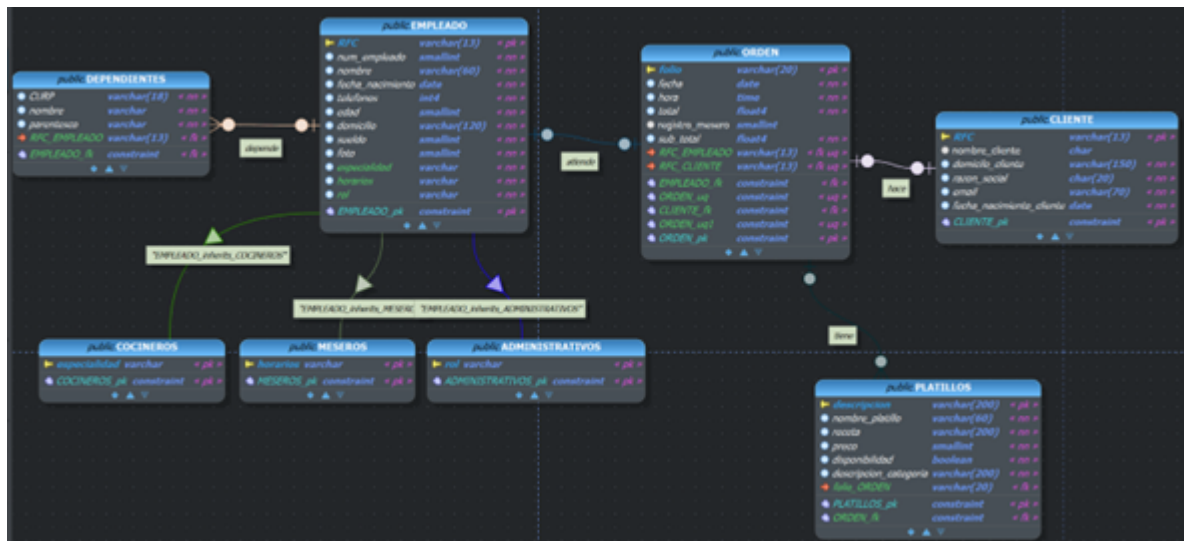
Aquí realizamos la relación entre dependientes y empleados donde se tuvo que hacer una generalización. Se tomó esta decisión porque los cocineros, meseros y administrativos contienen la misma información, por lo que fue más fácil crear un super tipo donde se juntan los atributos que tienen en común y que los subtipos tengan sus atributos únicos de la siguiente manera:



Ahora relacionamos la orden con el cliente. Pero dentro de la orden se apuntan algunos platillos que pueden ser bebidas o platillos.



Terminado el MER hicimos la implementación en PGmodeler del modelo relacional donde se declararon los tipos de datos que usaremos en cada atributo del modelo físico:



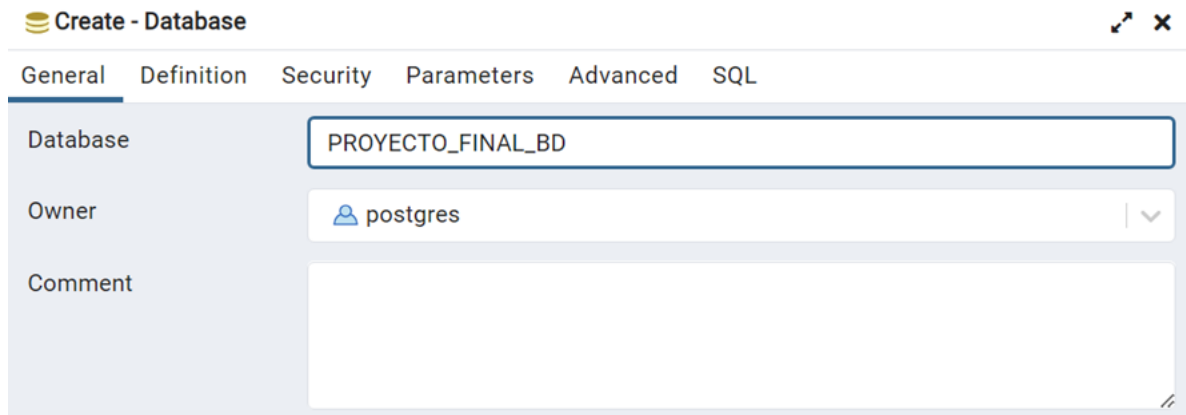
Tuvimos un problema al crear la generalización, de manera que adaptamos el modelo con los “inheritance”



4. Implementación

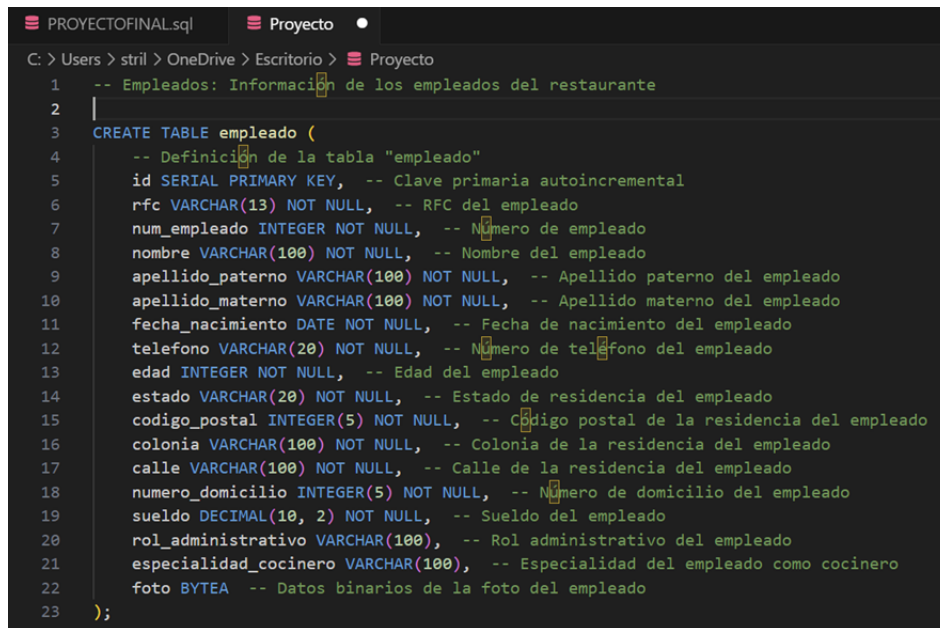
- Comenzamos por crear la base de datos en PGAdmin 4:

Owner: Postgres
Encoding UTF-8



The screenshot shows the 'Create - Database' dialog box in PGAdmin 4. The 'General' tab is selected. The 'Database' field contains 'PROYECTO_FINAL_BD'. The 'Owner' field shows a user icon and 'postgres'. The 'Comment' field is empty. The tabs at the top are General, Definition, Security, Parameters, Advanced, and SQL.

- Una vez creada la base de datos, debemos insertar el número de tablas con sus correspondientes atributos tal como se desarrolló en el Modelo Relacional.



The screenshot shows a SQL script in a code editor. The script is for creating a table named 'empleado'. It includes comments in Spanish describing the table and its attributes. The attributes are: id (SERIAL PRIMARY KEY), rfc (VARCHAR(13) NOT NULL), num_empleado (INTEGER NOT NULL), nombre (VARCHAR(100) NOT NULL), apellido_paterno (VARCHAR(100) NOT NULL), apellido_materno (VARCHAR(100) NOT NULL), fecha_nacimiento (DATE NOT NULL), telefono (VARCHAR(20) NOT NULL), edad (INTEGER NOT NULL), estado (VARCHAR(20) NOT NULL), codigo_postal (INTEGER(5) NOT NULL), colonia (VARCHAR(100) NOT NULL), calle (VARCHAR(100) NOT NULL), numero_domicilio (INTEGER(5) NOT NULL), sueldo (DECIMAL(10, 2) NOT NULL), rol_administrativo (VARCHAR(100)), especialidad_cocinero (VARCHAR(100)), and foto (BYTEA).

```
1  -- Empleados: Información de los empleados del restaurante
2
3  CREATE TABLE empleado (
4      -- Definición de la tabla "empleado"
5      id SERIAL PRIMARY KEY, -- Clave primaria autoincremental
6      rfc VARCHAR(13) NOT NULL, -- RFC del empleado
7      num_empleado INTEGER NOT NULL, -- Número de empleado
8      nombre VARCHAR(100) NOT NULL, -- Nombre del empleado
9      apellido_paterno VARCHAR(100) NOT NULL, -- Apellido paterno del empleado
10     apellido_materno VARCHAR(100) NOT NULL, -- Apellido materno del empleado
11     fecha_nacimiento DATE NOT NULL, -- Fecha de nacimiento del empleado
12     telefono VARCHAR(20) NOT NULL, -- Número de teléfono del empleado
13     edad INTEGER NOT NULL, -- Edad del empleado
14     estado VARCHAR(20) NOT NULL, -- Estado de residencia del empleado
15     codigo_postal INTEGER(5) NOT NULL, -- Código postal de la residencia del empleado
16     colonia VARCHAR(100) NOT NULL, -- Colonia de la residencia del empleado
17     calle VARCHAR(100) NOT NULL, -- Calle de la residencia del empleado
18     numero_domicilio INTEGER(5) NOT NULL, -- Número de domicilio del empleado
19     sueldo DECIMAL(10, 2) NOT NULL, -- Sueldo del empleado
20     rol_administrativo VARCHAR(100), -- Rol administrativo del empleado
21     especialidad_cocinero VARCHAR(100), -- Especialidad del empleado como cocinero
22     foto BYTEA -- Datos binarios de la foto del empleado
23 );
```

Extra a la documentación del proyecto, fue necesario agregar nuevos atributos como los apellidos paternos, y la información del domicilio como: estado, código postal, colonia, calle, NumeroDomicilio.

De igual manera como en el MR se hizo una generalización donde asignamos un rol y una especialidad como fue el caso del cocinero. Para estos datos ocupamos un tipo varchar limitando el número de caracteres para tener un mayor control de la inserción de datos.

- Continuamos con la creación de tablas que mantienen los tipos del Modelo Relacional.

```
-- Dependientes: Información de los dependientes de los empleados
CREATE TABLE dependientes (
    id SERIAL PRIMARY KEY,
    curp VARCHAR(18) NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    parentesco VARCHAR(50) NOT NULL,
    empleado_id INTEGER NOT NULL,
    FOREIGN KEY (empleado_id) REFERENCES empleados(id)
);

-- Platos: Información de los platos que ofrece el restaurante
CREATE TABLE platos (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    descripcion VARCHAR(200) NOT NULL,
    receta TEXT NOT NULL,
    precio DECIMAL(10, 2) NOT NULL,
    disponible BOOLEAN NOT NULL,
    categoria VARCHAR(50) NOT NULL
);

-- Bebidas: Información de las bebidas que ofrece el restaurante
CREATE TABLE bebidas (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    descripcion VARCHAR(200) NOT NULL,
    precio DECIMAL(10, 2) NOT NULL,
    disponible BOOLEAN NOT NULL,
    categoria VARCHAR(50) NOT NULL
);

-- Ordenes: Información de las ordenes realizadas en el restaurante
CREATE TABLE ordenes (
    id SERIAL PRIMARY KEY,
    folio VARCHAR(50) NOT NULL,
    fecha_hora TIMESTAMP NOT NULL,
    cantidad_total DECIMAL(10, 2) NOT NULL,
    mesero_id INTEGER NOT NULL,
    FOREIGN KEY (mesero_id) REFERENCES empleados(id)
);
```



```

-- Relación Platillos-Ordenes: Tabla intermedia para relacionar platillos y ordenes
CREATE TABLE platillos_ordenes (
    id SERIAL PRIMARY KEY,
    platillo_id INTEGER NOT NULL,
    orden_id INTEGER NOT NULL,
    cantidad INTEGER NOT NULL,
    precio_total DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (platillo_id) REFERENCES platillos(id),
    FOREIGN KEY (orden_id) REFERENCES ordenes(id)
);

-- Relación Bebidas-Ordenes: Tabla intermedia para relacionar bebidas y ordenes
CREATE TABLE bebidas_ordenes (
    id SERIAL PRIMARY KEY,
    bebida_id INTEGER NOT NULL,
    orden_id INTEGER NOT NULL,
    cantidad INTEGER NOT NULL,
    precio_total DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (bebida_id) REFERENCES bebidas(id),
    FOREIGN KEY (orden_id) REFERENCES ordenes(id)
);

-- Clientes: Información de los clientes que solicitan factura
CREATE TABLE clientes (
    id SERIAL PRIMARY KEY,
    rfc VARCHAR(13) NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    domicilio VARCHAR(200) NOT NULL,
    razon_social VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    fecha_nacimiento DATE NOT NULL
);

-- Puestos: Información de los distintos puestos que pueden tener los empleados
CREATE TABLE puestos (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(200) NOT NULL
);

```

```

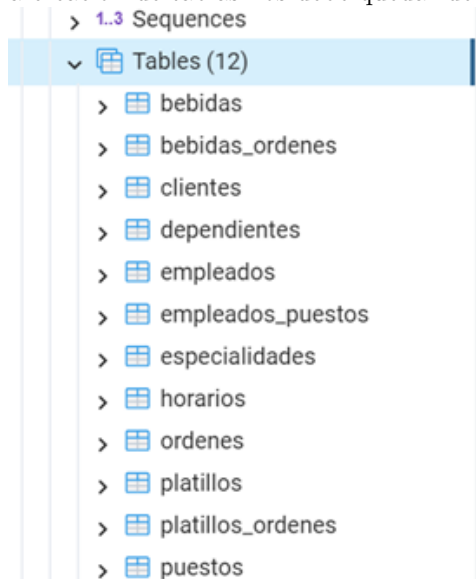
-- Horarios: Información de los horarios de trabajo de los meseros
CREATE TABLE horarios (
    id SERIAL PRIMARY KEY,
    mesero_id INTEGER NOT NULL,
    hora_entrada TIME NOT NULL,
    hora_salida TIME NOT NULL,
    FOREIGN KEY (mesero_id) REFERENCES empleados(id)
);

-- Especialidad: Información sobre la especialidad de los cocineros
CREATE TABLE especialidades (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(200) NOT NULL
);

-- Crear la tabla intermedia Empleados_Puestos
CREATE TABLE empleados_puestos (
    id SERIAL PRIMARY KEY,
    empleado_id INTEGER NOT NULL,
    puesto_id INTEGER NOT NULL,
    FOREIGN KEY (empleado_id) REFERENCES empleados(id),
    FOREIGN KEY (puesto_id) REFERENCES puestos(id)
);

```

- Una vez concluida la creación de tablas nos debe quedar de la siguiente manera:



La creación de estos índices está orientada a mejorar el rendimiento de las consultas que involucren las columnas especificadas en las tablas correspondientes. Sin embargo, es importante destacar que la creación de índices puede tener un impacto en el rendimiento de las operaciones de escritura (inserciones, actualizaciones y eliminaciones), por lo que se debe considerar cuidadosamente el equilibrio entre consultas más rápidas y el costo asociado con las operaciones de escritura.

```
-- Crear el índice en la tabla Dependientes
CREATE INDEX idx_dependientes_employado_id ON dependientes(employado_id);

-- Crear los índices en la tabla Ordenes
CREATE INDEX idx_ordenes_mesero_id ON ordenes(mesero_id);

-- Crear los índices en la tabla Horarios
CREATE INDEX idx_horarios_mesero_id ON horarios(mesero_id);
```

La función actualizar totales se utiliza como un disparador para actualizar automáticamente los totales en las tablas PlatosOrdenes y BebidasOrdenes cuando se insertan nuevos registros en una tabla de órdenes (ordenes).

```
153 -- Actualizar Totales Function: Actualiza los totales de las ordenes y productos al insertar nuevos registros
154 CREATE OR REPLACE FUNCTION actualizar_totales() RETURNS TRIGGER AS $$
155 BEGIN
156     IF NEW.platillo_id IS NOT NULL THEN
157         -- Actualizar el precio total de los platos en platos_ordenes
158         UPDATE platos_ordenes SET precio_total = NEW.cantidad * platos.precio
159         FROM platos WHERE platos.id = NEW.platillo_id AND platos.disponible = TRUE;
160     ELSE
161         -- Actualizar el precio total de las bebidas en bebidas_ordenes
162         UPDATE bebidas_ordenes SET precio_total = NEW.cantidad * bebidas.precio
163         FROM bebidas WHERE bebidas.id = NEW.bebida_id AND bebidas.disponible = TRUE;
164     END IF;
165
166     -- Actualizar la cantidad total en la orden
167     UPDATE ordenes SET cantidad_total = COALESCE(
168         (SELECT SUM(precio_total) FROM platos_ordenes WHERE orden_id = NEW.orden_id),
169         0)
170     +
171     COALESCE(
172         (SELECT SUM(precio_total) FROM bebidas_ordenes WHERE orden_id = NEW.orden_id),
173         0)
174     WHERE id = NEW.orden_id;
175
176     -- Verificar si algún producto no está disponible
177     IF NOT FOUND THEN
178         RAISE EXCEPTION 'Producto no disponible';
179     END IF;
180
181     RETURN NEW;
182 END;
183 $$ LANGUAGE plpgsql;
```

Cada vez que se realiza una inserción en las tablas PlatosOrdenes o BebidasOrdenes, se activará el disparador correspondiente (ActualizarTotalesPlatos o ActualizarTotalesBebidas). Estos disparadores, a su vez, ejecutan la función ActualizarTotales(), que se encarga de actualizar los totales en las tablas PlatosOrdenes y BebidasOrdenes y realizar otras verificaciones y actualizaciones necesarias según la lógica definida en la función ActualizarTotales().

```

185  -- Triggers
186  CREATE TRIGGER actualizar_totales_platillos
187  AFTER INSERT ON platillos_ordenes
188  FOR EACH ROW
189  EXECUTE FUNCTION actualizar_totales();
190
191  CREATE TRIGGER actualizar_totales_bebidas
192  AFTER INSERT ON bebidas_ordenes
193  FOR EACH ROW
194  EXECUTE FUNCTION actualizar_totales();
195

```

1. Nombre de la Vista: VistaFactura

2. Estructura de la Vista:

- a) La vista contiene varias columnas seleccionadas de diferentes tablas.
- b) o.folio, o.fechaHora, o.antidadTotal, y o.meseroId provienen de la tabla ordenes.
- c) e.nombre proviene de la tabla empleados y se renombra como NombreMesero.
- d) COALESCE(p.nombre, b.nombre) combina el nombre de un platillo (p.nombre) o una bebida (b.nombre) dependiendo de la disponibilidad del platillo o bebida en la orden
- e) COALESCE(po.cantidad, bo.cantidad) combina la cantidad de un platillo (po.cantidad) o una bebida (bo.cantidad) dependiendo de la disponibilidad del platillo o bebida en la orden.
- f) COALESCE(po.precioTotal, bo.precioTotal) combina el precio total de un platillo (po.precioTotal) o una bebida (bo.precioTotal) dependiendo de la disponibilidad del platillo o bebida en la orden.

1. Estructura del SELECT:

- a) Se utiliza un conjunto de instrucciones SELECT para seleccionar las columnas de diferentes tablas y combinarlas en una vista consolidada.
- b) Se utilizan operaciones de JOIN y LEFT JOIN para unir las tablas ordenes, empleados, PlatillosOrdenes, platillos, BebidasOrdenes, y bebidas en función de las claves primarias y foráneas.

```

6 -- Vista de Factura: Contiene información relevante para simular una factura
7 CREATE VIEW vista_factura AS
8 SELECT
9     o.folio,
10    o.fecha_hora,
11    o.cantidad_total,
12    o.mesero_id,
13    e.nombre AS nombre_mesero,
14    COALESCE(p.nombre, b.nombre) AS nombre_producto,
15    COALESCE(po.cantidad, bo.cantidad) AS cantidad_producto,
16    COALESCE(po.precio_total, bo.precio_total) AS precio_total_producto
17 FROM ordenes o
18 JOIN empleados e ON o.mesero_id = e.id
19 LEFT JOIN platillos_ordenes po ON o.id = po.orden_id
20 LEFT JOIN platillos p ON po.platillo_id = p.id
21 LEFT JOIN bebidas_ordenes bo ON o.id = bo.orden_id
22 LEFT JOIN bebidas b ON bo.bebida_id = b.id;

```

Esta vista VistaFactura parece estar diseñada para proporcionar información relevante para generar facturas. Combina datos de órdenes, meseros, platillos y bebidas en una estructura que facilita la obtención de detalles necesarios para simular una factura. La utilización de LEFT JOIN permite que la vista incluya órdenes incluso si no tienen platillos o bebidas asociados. Además, la función COALESCE maneja casos donde, por ejemplo, un platillo o bebida puede estar ausente en una orden. En esos casos, la columna correspondiente mostrará NULL y se puede gestionar en la lógica de facturación según sea necesario.

Debemos alterar una tabla, entonces creamos:

```

213
214 -- Añadir columna cliente_id a Ordenes
215 ALTER TABLE ordenes ADD COLUMN cliente_id INTEGER;
216

```

Después de ejecutar este comando, la tabla ordenes incluirá una nueva columna llamada ClienteId que puede ser utilizada para almacenar el identificador del cliente asociado a una orden.

Ya por último creamos una vista que nos permita ver en las facturas las fechas y total de las órdenes.

```

218 -- Vista de Factura: Proporciona información necesaria para asemejarse a una factura y una orden
219 CREATE VIEW factura AS
220 SELECT
221     CONCAT('ORD-', LPAD(CAST(ordenes.id AS VARCHAR), 3, '0')) AS folio,
222     ordenes.fecha_hora,
223     ordenes.cantidad_total,
224     CONCAT(empleados.nombre, ' ', empleados.apellido_paterno, ' ', empleados.apellido_materno) AS mesero
225 FROM
226     ordenes
227 JOIN empleados ON ordenes.mesero_id = empleados.id;
228
229

```

5. Presentación

El sitio web donde se despliega la información de los empleados fue construido con las herramientas ASP.NET CORE 6.0 y el framework ORM Entity Framework 6, junto con la utilización del paquete Npgsql.EntityFrameworkCore.PostgreSQL para lograr la conexión a una base de datos que resida en un servidor de Postgres.

6. Conclusiones

Gómez Enríquez Agustín:

El diseño de una base de datos para un restaurante es un proceso complejo que requiere un profundo conocimiento de los requerimientos del negocio.

El diseño presentado en este documento cumple con todos los requerimientos planteados y, además, cumple con los principios de diseño fundamentales para una base de datos. La base de datos está organizada de forma eficiente y permite almacenar y gestionar la información de forma eficaz.

La implementación de la base de datos permitirá al restaurante digitalizar sus procesos y mejorar la gestión de su información. Esto puede suponer un importante ahorro de tiempo y recursos, así como una mejora en la calidad del servicio al cliente.

Walls Chávez Luis Fernando:

La implementación de una base de datos para un restaurante puede suponer una importante mejora en la gestión del negocio. La base de datos puede proporcionar información valiosa sobre el funcionamiento del restaurante, como los ingresos, los platillos más vendidos y las horas pico de mayor actividad.

En el caso del restaurante descrito en este documento, la implementación de la base de datos permitirá al restaurante:

1. Automatizar los procesos de gestión de empleados, platillos, órdenes y clientes.
2. Mejorar la eficiencia de los procesos de facturación y cobro.
3. Obtener información valiosa para la toma de decisiones.

Madera Fuente Jose Angel

En el caso del restaurante descrito en este documento, el diseño presentado es flexible y escalable. Esto significa que la base de datos puede adaptarse a los cambios en el negocio, como el aumento del número de empleados, el lanzamiento de nuevos platillos o la expansión del restaurante a nuevas ubicaciones.

La flexibilidad y la escalabilidad de la base de datos son importantes para garantizar que el restaurante pueda aprovechar al máximo las ventajas de la digitalización. La base de datos debe ser una herramienta que ayude al restaurante a crecer y prosperar.