
Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
Bases de Datos 1

***Proyecto 2***

Desarrollo y Consulta de Bases de Datos Operativas

Carlos Magaña
Jorge Chupina
Derick Delva
Jose Alejandro Anton

**Docente:**
Mario Barrientos

**Fecha:**
Guatemala, septiembre 2024

## Contexto del Problema a Modelar

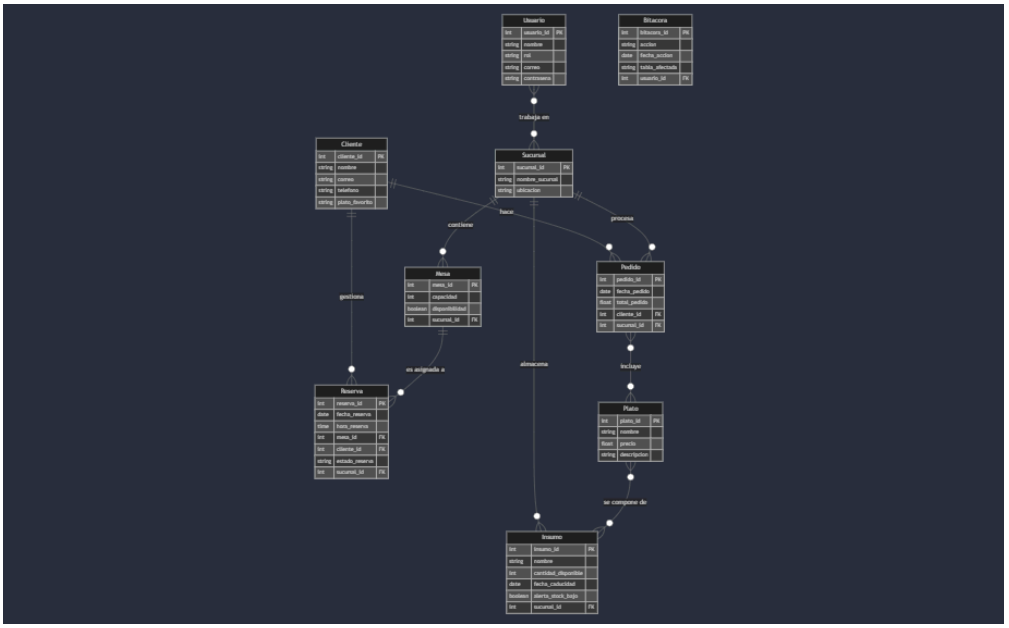
Los restaurantes a menudo enfrentan desafíos en la gestión de reservas, la optimización del uso de mesas, el control del inventario de alimentos y bebidas, y el seguimiento de las preferencias y comportamientos de los clientes. Para grandes cadenas de restaurantes, sin un sistema centralizado, se dificulta la toma de decisiones informadas, lo que puede llevar a pérdidas financieras, desperdicio de alimentos y una experiencia de cliente deficiente.

## Objetivo General del Proyecto y Problemática

Se busca utilizar de tecnologías de bases de datos para la creación y carga de modelos de datos, con el objetivo de utilizar lenguaje SQL para la gestión eficiente de restaurantes, enfocándose en reservas de mesas, control de inventario de insumos, y seguimiento de clientes para mejorar la toma de decisiones y la eficiencia operativa.

## Solución de la Problemática

Para resolver la problemática presente, se decidió utilizar Python para el desarrollo del sistema al igual que una conexión a la base de datos en Postgres.
Primero, se realizó un diagrama entidad relación que cumpla con las necesidades del proyecto y pueda modelar perfectamente la situación que estamos resolviendo. (Imagen de alta calidad adjunta en los archivos del proyecto.)




Posteriormente, decidimos realizar la creación de tablas en pgadmin. Todo esto mediante el documento cuyo nombre es Creación_tablas.sql.

**Formato de la Base de Datos:**

**Tabla: bitacora**

Descripción: Registra las acciones realizadas en el sistema.

Columnas:

**bitacora_id**: integer NOT NULL (Clave primaria, autoincremental)

**accion**: character varying(120) (Descripción de la acción realizada)

**fecha_accion**: date (Fecha en la que se realizó la acción)

**tabla_afectada**: character varying(50) (Nombre de la tabla afectada por la acción)

**Tabla: cliente**

Descripción: Almacena los datos de los clientes del restaurante.

Columnas:

**cliente_id:** integer NOT NULL (Clave primaria, autoincremental)

**nombre:** character varying(50) (Nombre del cliente)

**correo:** character varying(50) (Correo electrónico único del cliente)

**telefono:** character varying(50) (Teléfono de contacto del cliente)

**plato_favorito:** character varying(50) (Plato preferido del cliente)

**Tabla: insumo**

Descripción: Almacena los insumos o ingredientes disponibles en cada sucursal.

Columnas:

**insumo_id**: integer NOT NULL (Clave primaria, autoincremental)

**nombre:** character varying(50) (Nombre del insumo)

**cantidad_disponible**: integer (Cantidad disponible del insumo)

**fecha_caducidad**: date (Fecha de caducidad del insumo)

**alerta_stock_bajo**: boolean (Indica si el stock está bajo)

**sucursal_id**: integer (Clave foránea que referencia la sucursal)

Restricciones:

**chk_cantidad_disponible:** Verifica que **cantidad_disponible** esté entre 0 y 100.

**Tabla: insumo_plato**

Descripción: Relaciona los insumos con los platos que los utilizan.

Columnas:

**insumo_id**: integer NOT NULL (Clave foránea que referencia el insumo)

**plato_id**: integer NOT NULL (Clave foránea que referencia el plato)

Llaves:

Primaria: **insumo_id**, **plato_id**

**Tabla: mesa**

Descripción: Almacena información sobre las mesas en las sucursales.

Columnas:

**mesa_id:** integer NOT NULL (Clave primaria, autoincremental)

**capacidad:** integer (Número de personas que caben en la mesa)

**disponibilidad:** boolean (Indica si la mesa está disponible)

**sucursal_id:** integer (Clave foránea que referencia la sucursal)

**Tabla: pedido**

Descripción: Almacena los pedidos realizados por los clientes.

Columnas:

**pedido_id:** integer NOT NULL (Clave primaria, autoincremental)

**fecha_pedido:** date (Fecha en la que se realizó el pedido)

**total_pedido:** float (Monto total del pedido)

**cliente_id:** integer (Clave foránea que referencia al cliente)

**sucursal_id:** integer (Clave foránea que referencia la sucursal)

**mesa_id:** integer (Clave foránea que referencia la mesa donde se hizo el pedido)

**Tabla: pedido_plato**

Descripción: Relaciona los platos pedidos en cada pedido.

Columnas:

**pedido_id:** integer NOT NULL (Clave foránea que referencia el pedido)

**plato_id:** integer NOT NULL (Clave foránea que referencia el plato)

Llaves:

Primaria: **pedido_id**, **plato_id**

**Tabla: plato**

Descripción: Almacena la información de los platos ofrecidos por el restaurante.

Columnas:

**plato_id:** integer NOT NULL (Clave primaria, autoincremental)

**nombre:** character varying(50) (Nombre del plato)

**precio:** float CHECK (precio >= 0) (Precio del plato)

**descripcion:** character varying(255) (Descripción del plato)

**Tabla: reserva**

Descripción: Almacena las reservas realizadas por los clientes para las mesas.

Columnas:

**reserva_id:** integer NOT NULL (Clave primaria, autoincremental)

**fecha_reserva:** date (Fecha de la reserva)

**mesa_id:** integer (Clave foránea que referencia la mesa reservada)

**cliente_id:** integer (Clave foránea que referencia al cliente que realizó la reserva)

**estado_reserva:** varchar(50) (Estado de la reserva: pendiente, confirmada, cancelada)

**sucursal_id:** integer (Clave foránea que referencia la sucursal donde se hizo la reserva)

**Tabla: sucursal**

Descripción: Almacena la información de las sucursales del restaurante.

Columnas:

**sucursal_id:** integer NOT NULL (Clave primaria, autoincremental)

**nombre_sucursal:** varchar(50) (Nombre de la sucursal)

**ubicacion:** varchar(50) (Dirección o ubicación de la sucursal)

Después, decidimos sobre el restaurante al cual ayudaremos y su menú para insertar datos importantes a la base de datos. Dicho menú será para todas las sedes para temas de simplicidad en los insumos y demás. Nombre de la Cadena de Restaurantes:

# BALBINOS PIZZERIA

**Pizza Menu**

1. **Margherita**
    - Ingredients: Tomato Sauce, Mozzarella, Basil, Bread
    - Price: $8.99
2. **Pepperoni**
    - Ingredients: Tomato Sauce, Mozzarella, Pepperoni, Bread
    - Price: $9.99
3. **BBQ Chicken**
    - Ingredients: BBQ Sauce, Grilled Chicken, Red Onions, Mozzarella, Bread
    - Price: $10.99
4. **Veggie**
    - Ingredients: Tomato Sauce, Mozzarella, Bell Peppers, Onions, Olives, Bread
    - Price: $9.49
5. **Hawaiian**
    - Ingredients: Tomato Sauce, Mozzarella, Ham, Pineapple, Bread
    - Price: $10.49
6. **Four Cheese**
    - Ingredients: Tomato Sauce, Mozzarella, Cheddar, Parmesan, Blue Cheese, Bread
    - Price: $11.49
7. **Meat Lover's**
    - Ingredients: Tomato Sauce, Mozzarella, Pepperoni, Sausage, Ham, Bacon, Bread
    - Price: $12.99
8. **Buffalo Chicken**
    - Ingredients: Buffalo Sauce, Grilled Chicken, Mozzarella, Red Onions, Bread
    - Price: $10.99
9. **Mexican Pizza**
    - Ingredients: Salsa, Mozzarella, Ground Beef, Jalapeños, Red Onions, Bread
    - Price: $11.49
10. **Spinach & Feta**
    - Ingredients: White Sauce, Mozzarella, Spinach, Feta Cheese, Bread
    - Price: $9.99
11. **Caprese**
    - Ingredients: Tomato Sauce, Mozzarella, Fresh Tomato Slices, Basil, Bread
    - Price: $9.49
12. **Truffle Mushroom**
    - Ingredients: White Sauce, Mozzarella, Truffle Oil, Mushrooms, Bread
    - Price: $12.49
13. **Pesto Chicken**
    - Ingredients: Pesto Sauce, Grilled Chicken, Mozzarella, Sun-Dried Tomatoes, Bread
    - Price: $11.49
14. **Greek Pizza**

- Ingredients: Tomato Sauce, Mozzarella, Feta Cheese, Olives, Red Onions, Bread
- Price: $10.49

**En base a este menú y demás decidimos realizar los inserts estándar de la base de datos.**

Al tener planteada la estructura, se inició con el desarrollo de los archivos en Python que correrán el sistema de gestión de restaurantes. Inicialmente se creó la conexión a la base de datos, el programa utiliza la biblioteca psycopg2 para conectarse a la base de datos PostgreSQL. A continuación, se creo el menú principal, para este, se estableció la gestión de usuarios:

El programa permite a los usuarios iniciar sesión y registrar nuevos usuarios. Para el inicio de sesión, se verifica la combinación de correo y contraseña para autenticar al usuario. Los nuevos usuarios pueden registrarse proporcionando su nombre, correo, contraseña y rol asignado.

Dependiendo del rol del usuario (mesero, administrador o gerente), se despliega un menú con opciones específicas. Cada usuario tiene acceso solo a las funcionalidades relevantes para su rol.

## Administrador

**Para el rol de administrador utilizamos el siguiente conjunto de funciones en forma de menú:**

- **gestion_de_insumos_administrador()**

Permite al administrador gestionar los insumos de una sucursal específica.

Muestra una lista de sucursales disponibles y permite al usuario seleccionar una. Utiliza consultas SQL para interactuar con la base de datos y actualizar o recuperar información sobre los insumos.

Ofrece un menú con opciones para:

a) Registrar nuevos insumos o actualizar la cantidad existente.

```
UPDATE insumo

SET cantidad_disponible = cantidad_disponible + %s

WHERE insumo_id = %s and sucursal_id = %s;
```

b) Visualizar todos los insumos de la sucursal.

```
SELECT * FROM insumo WHERE sucursal_id = %s;
```

c) Ver insumos con stock bajo (menos de 10 unidades).

SELECT * FROM insumo WHERE sucursal_id = %s AND cantidad_disponible < 10;

- **customer_history()**

Obtener información del cliente:

SELECT nombre, correo, telefono, plato_favorito

FROM Cliente

WHERE cliente_id = %s;

Obtener reservaciones o pedidos del cliente.

Reservaciones:

SELECT R.fecha_reserva, M.mesa_id, R.estado_reserva, S.nombre_sucursal

FROM Reserva R

JOIN Mesa M ON R.mesa_id = M.mesa_id

JOIN Sucursal S ON R.sucursal_id = S.sucursal_id

WHERE R.cliente_id = %s;

Pedidos:

SELECT P.fecha_pedido, P.total_pedido, S.nombre_sucursal

FROM Pedido P

JOIN Sucursal S ON P.sucursal_id = S.sucursal_id

WHERE P.cliente_id = %s;

- **reporteria_administrador()**

Genera varios reportes para una sucursal específica. Permite al usuario seleccionar una sucursal de una lista, y después genera los siguientes reportes:

a) Top 10 de platos más vendidos.

SELECT P.nombre, COUNT(PP.plato_id) AS cantidad_vendida

```
FROM Pedido_Plato PP
JOIN Plato P ON PP.plato_id = P.plato_id
JOIN Pedido PD ON PP.pedido_id = PD.pedido_id
WHERE PD.sucursal_id = %s
GROUP BY P.nombre
ORDER BY cantidad_vendida DESC
LIMIT 10;
```

b) Top 10 de clientes más frecuentes.

```
SELECT C.nombre, COUNT(R.cliente_id) AS cantidad_visitas
FROM Reserva R
JOIN Cliente C ON R.cliente_id = C.cliente_id
WHERE R.sucursal_id = %s
GROUP BY C.nombre
ORDER BY cantidad_visitas DESC
LIMIT 10;
```

c) Top 5 de clientes con mayores reservas y sus platos favoritos.

```
SELECT C.nombre, COUNT(R.reserva_id) AS cantidad_reservas, C.plato_favorito
FROM Reserva R
JOIN Cliente C ON R.cliente_id = C.cliente_id
WHERE R.sucursal_id = %s
GROUP BY C.nombre, C.plato_favorito
ORDER BY cantidad_reservas DESC
LIMIT 5;
```

d) Reporte de insumos a punto de terminarse o caducar.

```
SELECT nombre, cantidad_disponible, fecha_caducidad

FROM Insumo

WHERE sucursal_id = %s AND (cantidad_disponible < 10 OR fecha_caducidad < NOW() + INTERVAL '1 month');
```

e) Comportamiento de la sucursal en términos de reservas y ventas.

```
SELECT S.nombre_sucursal, COUNT(R.reserva_id) AS cantidad_reservas, SUM(P.total_pedido) AS total_ventas

FROM Sucursal S

LEFT JOIN Reserva R ON S.sucursal_id = R.sucursal_id

LEFT JOIN Pedido P ON S.sucursal_id = P.sucursal_id

WHERE S.sucursal_id = %s

GROUP BY S.nombre_sucursal

ORDER BY cantidad_reservas DESC, total_ventas DESC;
```

- **control_de_cambios_administrador()**

Obtiene los cambios recientes.

```
SELECT accion, fecha_accion, tabla_afectada

FROM Bitacora

WHERE tabla_afectada IN (

    'Sucursal', 'Cliente', 'Mesa', 'Reserva', 'Pedido', 'Plato', 'Insumo'

)

ORDER BY fecha_accion DESC;
```

## Gerente

**Para el rol de gerente utilizamos el siguiente conjunto de funciones en forma de menú:**

- **agregar_cliente():**

Agrega un nuevo cliente al conjunto de restaurantes lo cual abre la posibilidad de realizar pedidos y reservas hacia ese cliente.

Se definieron funciones para insercion de datos a la base.

```
def agregar_cliente():

nombre = input('Ingrese su nombre: ')

correo = input('Ingrese su correo: ')

telefono = input('Ingrese su teléfono: ')

plato_favorito = input('Ingrese su plato favorito: ')
```

Se añade la información en Python y se ingresa a postgres mediante esta consulta:

```
INSERT INTO cliente (nombre, correo, telefono, plato_favorito)

VALUES (%s, %s, %s, %s);
```

- **agregar_pedido():** Permite registrar un nuevo pedido hecho por un cliente en una sucursal específica. También gestiona la asignación de una mesa (si es requerida) y la vinculación de platos a dicho pedido.

Insertar nuevo pedido:

```
INSERT INTO pedido (fecha_pedido, total_pedido, cliente_id, sucursal_id, mesa_id)

VALUES (%s, %s, %s, %s, %s) RETURNING pedido_id;
```

Mesas disponibles:

```
SELECT mesa_id

FROM mesa

WHERE disponibilidad = TRUE AND sucursal_id = %s;
```

Agregar platos al pedido:

```
INSERT INTO pedido_plato (pedido_id, plato_id)

VALUES (%s, %s);
```

- **gestionar_reservas():** Contiene dos acciones principales: crear o finalizar reservas. En la creación de reservas, se verifica la disponibilidad de mesas en una sucursal para un cliente en una fecha específica y se registra la reserva si hay mesas libres. La función también permite finalizar una reserva, actualizando su estado a 'FINALIZADA'.

Obtener mesas disponibles:

```
SELECT mesa_id

FROM mesa

WHERE disponibilidad = TRUE and sucursal_id = %s;
```

Crear nueva reserva:

```
INSERT INTO reserva (fecha_reserva, mesa_id, cliente_id, estado, sucursal_id)

VALUES (%s, %s, %s, 'VIGENTE', %s) RETURNING reserva_id;
```

Finalizar reserva:

```
UPDATE reserva

SET estado = 'FINALIZADA'

WHERE reserva_id = %s AND sucursal_id = %s;
```

- **gestionar_mesas():** Se pueden visualizar las mesas ocupadas en una sucursal específica y desbloquear mesas cuando estas se liberen.

Visualizar mesas ocupadas:

```
SELECT mesa_id

FROM mesa

WHERE sucursal_id = %s AND disponibilidad = FALSE;
```

Desbloquear mesas:

```
UPDATE mesa

SET disponibilidad = TRUE

WHERE sucursal_id = %s AND mesa_id = %s;
```

- **visualizar_clientes():** Muestra los datos de los clientes registrados en el sistema. Puede ser utilizada para obtener información de los clientes, como nombre, contacto o historial de pedidos y reservas.

SELECT * FROM cliente WHERE nombre = %s AND correo = %s

- **gestion_de_insumos_gerente():** Ofrece un submenú para gestionar insumos, incluyendo registrar nuevos insumos, visualizar todos los insumos de la sucursal, y ver insumos con stock bajo. (Mismas consulta de SQL que en el menú de gestión de gestion_de_insumos_administrador())

- **control_de_cambios_gerente():** Muestra un registro de cambios recientes en varias tablas relacionadas con la sucursal del gerente. control_de_cambios_administrador() y control_de_cambios_gerente() usan la misma consulta para obtener los cambios recientes de la bitácora.

- **reporteria_gerente():** Genera varios reportes, las funciones reporteria_administrador() y reporteria_gerente() utilizan exactamente las mismas consultas.

## Mesero

**Para el rol de mesero utilizamos el siguiente conjunto de funciones en forma de menú:**

- **agregar_cliente():** Misma función que en gerente.

- **agregar_pedido():** Misma función que en gerente.

- **gestionar_reservas():** Permite crear o finalizar reservas.

Obtener mesa disponible:

SELECT mesa_id

FROM mesa

WHERE disponibilidad = TRUE and sucursal_id = %s;

Crear nueva reserva:

INSERT INTO reserva (fecha_reserva, mesa_id, cliente_id, estado, sucursal_id)

VALUES (%s, %s, %s, 'VIGENTE', %s) RETURNING reserva_id;

Finalizar reserva:

UPDATE reserva

SET estado = 'FINALIZADA'

WHERE reserva_id = %s AND sucursal_id = %s;

- **gestionar_mesas():** Misma función que en gerente.

- **visualizar_clientes():** Muestra los datos de los clientes registrados en el sistema.

## Funciones y Triggers

## Creación de triggers (ver triggers_proyecto2.sql).

**bloquear_mesa_por_reserva:**

- Momento: Después de insertar una nueva reserva.
- Acción: Bloquea (marca como ocupada) una mesa cuando se inserta una nueva reserva en estado "VIGENTE", cambiando su disponibilidad a FALSE.

**desbloquear_mesa_por_finalizacion:**

- Momento: Después de actualizar una reserva.
- Acción: Desbloquea una mesa (la marca como disponible) cuando el estado de la reserva se actualiza a "FINALIZADA". Registra la acción en la bitacora.

**bloquear_mesa_por_pedido:**

- Momento: Después de insertar un nuevo pedido.
- Acción: Bloquea una mesa cuando se realiza un pedido asociado a esa mesa (si mesa_id no es nulo). Si el pedido es a domicilio, registra la acción en la bitacora sin afectar la mesa.

**insertar_bitacora_finalizacion_reserva:**

- Momento: Después de actualizar una reserva.

- Acción: Registra en la bitacora cuando una reserva cambia su estado a "FINALIZADA", manteniendo un historial de reservas completadas.

**actualizar_insumos:**

- Momento: No se especifica el trigger explícito en el código, pero presumiblemente asociado a la inserción de pedidos.
- Acción: Actualiza la cantidad disponible de insumos utilizados en los platos de cada pedido, emitiendo alertas cuando la cantidad baja de 16 unidades.

**trigger_bitacora_insumos:**

- Momento: Después de actualizar un insumo.
- Acción: Registra en la bitacora cualquier incremento en la cantidad disponible de insumos, para llevar un control del inventario.

**trigger_desbloquear_mesa:**

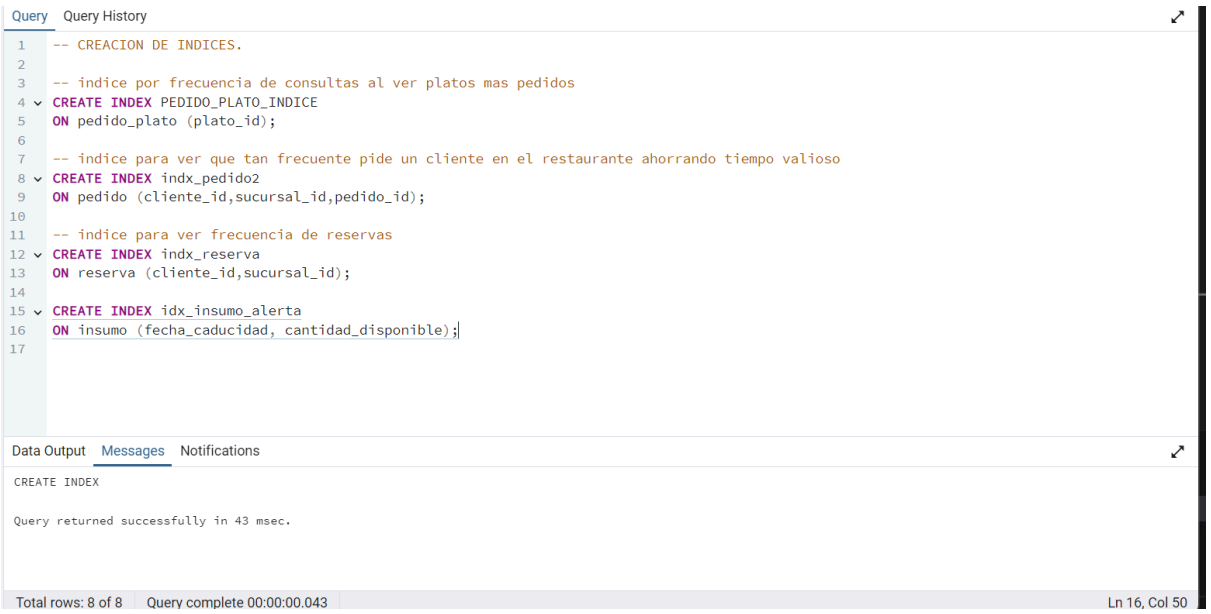
- Momento: Después de actualizar la disponibilidad de una mesa.
- Acción: Registra en la bitacora cuando una mesa es desbloqueada (su disponibilidad cambia a TRUE), indicando que la mesa está nuevamente disponible.

## Supuestos de vital importancia:

- Las mesas tienen la misma cantidad de personas.
- Los restaurantes tienen misma cantidad de mesas
- Cada plato de comida solo consume un paquete de cada insumo relacionado
- Por problemas técnicos la sucursal se volverá a preguntar posterior a cada inicio de sesión.
- Los datos insertados al sistema serán correctos sin ninguna falla en el tipo de dato.
- Las reservas se harán en un día anterior ya que, al hacerla la mesa se habilitará para todos.
- Se realizarán pedidos a domicilio mediante la identificación de mesa = null.
- El precio del pedido será calculado antes de solicitar los platos a comer.

## Los índices implementados son los siguientes (Ver Indices_proyecto2.sql)

En cuanto al desarrollo de las siguientes problemática de optimización de tiempos de consultas y demás decidimos implementar ciertos índices de forma estratégica con tal de que se facilite el acceso a la información evitando esperas muy largas entre consultas.




**Indice 1:**

Básicamente decidimos realizar un indice en pedido_plato especificamente plato_id ya que para cada pedido obtendremos una cantidad masiva de datos en pedido_plato por lo tanto, será bastante útil a la hora de buscar el plato más vendido de la cadena de restaurantes. Además visualizamos un patrón bastante grande en donde este atributo está presente en bastantes querys para ser específicos en cláusulas de join, group by y where.

**Índice 2**

Básicamente decidimos realizar un índice en las columnas cliente_id, sucursal_id y pedido_id porque cada vez que consultemos los pedidos de un cliente en una sucursal específica, este índice mejorará significativamente el rendimiento. Como esperamos un gran volumen de pedidos en cada sucursal, este índice nos permitirá agrupar y filtrar los datos más rápidamente. Además, estas columnas suelen aparecer en varias consultas que involucran análisis por cliente o sucursal, particularmente en cláusulas WHERE y JOIN, lo que hace que sea esencial para optimizar el acceso.

**Índice 3**

Decidimos implementar un índice sobre cliente_id y sucursal_id en la tabla reserva porque las consultas que buscan la frecuencia de reservas por cliente en una sucursal en particular son comunes. Este índice reduce el tiempo de búsqueda cuando queremos saber cuántas veces un cliente ha reservado en una sucursal específica. Además, observamos que estas columnas son frecuentemente usadas en JOIN y filtros WHERE, por lo que el índice mejorará tanto las consultas sobre la cantidad de reservas como sobre la eficiencia de agrupamiento en reportes de comportamiento de clientes.

**Índice 4**

Optamos por crear este índice sobre fecha_caducidad y cantidad_disponible porque las consultas que buscan insumos próximos a caducar o con bajo stock son críticas para la

gestión del inventario. Estas dos columnas son esenciales en reportes periódicos que monitorean el estado de los insumos y, al combinarlas en un solo índice, podemos optimizar tanto las búsquedas de insumos a punto de caducar como los que están cerca de quedarse sin stock. Dado que estas columnas aparecen frecuentemente en consultas WHERE, el índice reducirá significativamente los tiempos de consulta en estas situaciones.

**NOTA:**

**Todo el trabajo se realizó por partes y está registrado en el siguiente github:**

**CarFAngM/uvg_BD (github.com)**