Base de datos

Peluquerías Paquita

DAVANTE | MEDAC
Carlos Piernas Jiménez
Acceso a Datos
José Ferrer Grau
28-11-2025

Índice

Introducción

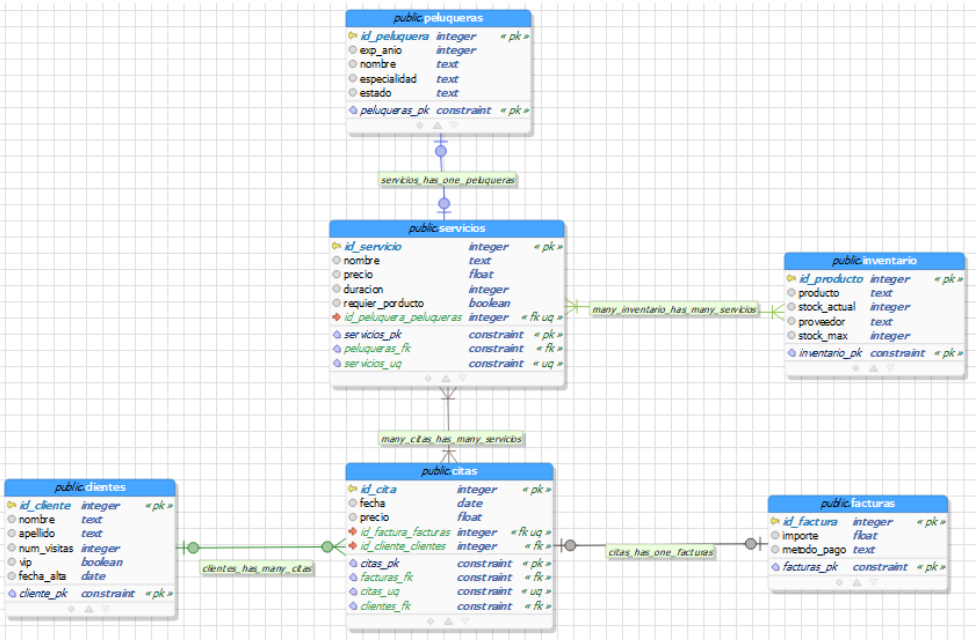
Como proyecto final de trimestre se nos ha pedido que hagamos una base de datos para una peluquería, teniendo unas entidades y atributos predeterminados, siendo estos:

- Clientes: datos personales, tipo de cliente (normal o VIP), número de visitas, fecha de alta, etc.
- Peluqueras: nombre, años de experiencia, especialidad (corte, tinte, peinado...), y estado (disponible, descansando, de baja).
- Servicios: nombre del servicio (lavado, corte, tinte, peinado...), precio, duración media y si requiere producto específico.
- Citas o atenciones: registro de cada cliente atendido, la peluquera asignada, los servicios realizados y el precio total.
- Inventario de productos: champús, tintes, lacas, etc. con su stock actual y proveedor.
- Facturas o tickets: documento asociados a cada atención con fecha, importe y método de pago.

Teniendo esto como base de la base de datos se ha construido todo lo siguiente.

Diagrama Relación - Entidad

Para el diagrama de relación - entidad se ha preparado a papel una idea de lo que se quería construir, luego se ha pasado esta idea a un diagrama completo hecho en el programa “pgModeler” que tendría el siguiente aspecto:



En este diagrama se muestran las siguientes relaciones entre entidades:

- **Peluqueras - Servicio:** Esta relación tiene una cardinalidad 1:1, ya que una peluquera hace un servicio.
- **Servicio - Inventario:** Esta relación tiene una cardinalidad N:M, ya que varios servicios pueden tener varios productos del inventario.
- **Servicio - Citas:** Esta relación tiene una cardinalidad N:M, ya que varios servicios pueden aparecer en varias citas.
- **Citas -Facturas:** Esta relación tiene una cardinalidad 1:1, ya que una factura pertenece a una cita.
- **Citas - Clientes:** Esta relación tiene una cardinalidad 1:1, ya que una cita pertenece a un solo cliente.

Las relaciones N:M crean la necesidad de generar tablas intermedias, siendo estas **citas_servicios** e **inventario_servicios**.

Cada atributo tiene un tipo de dato adecuado, siendo los siguientes:

- **Clientes:**
    - id_cliente (PK): integer.
    - nombre: text.
    - apellido: text.
    - vip: boolean.
    - fecha de alta: date.
- **Peluqueras:**
    - id_peluquera (PK): integer.
    - exp_anio: integer.
    - nombre: text.
    - especialidad: text.
    - estado: text.
- **Servicios:**
    - id_servicio (PK): integer.
    - nombre: text.
    - precio: double precision.
    - duracion: integer.
    - requiere_producto: boolean.
    - id_peluquera_peluqueras (FK): integer.

- **Citas:**
    - id_cita (PK): integer.
    - fecha: date.
    - precio: double precision.
    - id_factura_facturas (FK): integer.
    - id_cliente_clientes (FK): integer.
- **Inventario:**
    - id_producto (PK): integer.
    - producto: text.
    - stock_actual: integer.
    - proveedor: text.
    - stock_max: integer.
- **Factura:**
    - id_factura: integer.
    - importe: double precision.
    - método_pago: text.
- **Citas_inventario:**
    - id_cita_citas (FK): integer.
    - id_servicio_servicios (FK): integer.
- **Inventario_servicios:**
    - id_producto_inventario (FK): integer.
    - id_servicio_servicios (FK): integer.

Creación de la base de datos.

Para la creación de la base de datos se han seguido los pasos presentados en proyectos anteriores, que sería copiar el código generado por “pgModeler” en el apartado de “Source” y pegarlo en la “Query tool” del programa “pgAdmin 4”, donde se ejecutará este código en lenguaje sql.
Una vez creada la base de datos se ha creado un programa de generación de datos. Este programa genera los datos en un archivo “.txt” con formato “csv” (los datos van separados por comas, sin espacios). Estos datos se introducen en la base de datos utilizando la función de importar en cada una de las tablas.

Para este proyecto se han generado 500 clientes, 2500 citas y facturas, 6 peluqueras, 15 productos y 10 servicios.

Índices.

En este proyecto también se pide la creación de índices de los campos más consultados. Se ha determinado que los campos más consultados serían las consultas que se hacen en la interfaz del proyecto de interfaces, y serían los siguientes:

1. **Índice para la consulta de clientes utilizada en las interfaces**

```
CREATE INDEX clientesInterfaz
ON clientes (id_cliente, nombre, apellido, num_visitas, vip, fecha_alta);
```

2. **Índice para la consulta de peluqueras utilizada en las interfaces**

```
CREATE INDEX peluquerasInterfaz
ON peluqueras (id_peluquera, exp_anio, nombre, especialidad, estado);
```

3. **Índice para la consulta de servicios utilizada en las interfaces**

```
CREATE INDEX serviciosInterfaz
ON servicios (id_servicio, nombre, precio, duracion, requier_porducto, id_peluquera_peluqueras);
```

4. **Índice para la consulta de productos utilizada en las interfaces**

```
CREATE INDEX inventarioInterfaz
ON inventario (id_producto, producto, stock_actual, proveedor, stock_max);
```

5. **Índice para la consulta de clientes vip utilizada en las interfaces**

```
CREATE INDEX clientes_vip
ON clientes (id_cliente, nombre, apellido, num_visitas, vip, fecha_alta)
WHERE vip = true;
```

6. **Índice para la consulta de productos con stock bajo utilizada en las interfaces**

```
CREATE INDEX stockBajoInterfaz
ON inventario (id_producto, producto, stock_actual, proveedor, stock_max)
WHERE stock_actual < 5;
```

Consultas

En el proyecto también se nos pide que hagamos las siguientes consultas:

1. Top de peluqueras más activas: mostrar las 3 peluqueras que más clientes han atendido en el último mes.

```
SELECT p.nombre, COUNT(c.id_cita) AS total_citas_ultimo_mes
FROM peluqueras p
JOIN servicios s ON p.id_peluquera = s.id_peluquera_peluqueras
JOIN citas_servicios cs ON s.id_servicio = cs.id_servicio_servicios
JOIN citas c ON cs.id_cita_citas = c.id_cita
WHERE c.fecha >= DATE '2025-12-30' - INTERVAL '1 month'
GROUP BY p.nombre
ORDER BY total_citas_ultimo_mes DESC
LIMIT 3;
```

	nombre text	total_citas_ultimo_mes bigint
1	JULIA	3
2	ALVARO	1
3	DRISS	1

2. Clientes frecuentes: listar los clientes que han acudido más de 5 veces durante el último trimestre.

```
SELECT cl.nombre, cl.apellido, COUNT(c.id_cita) AS total_visitas_trimestre
FROM clientes cl
JOIN citas c ON cl.id_cliente = c.id_cliente_clientes
WHERE c.fecha >= DATE '2025-12-30' - INTERVAL '3 months'
GROUP BY cl.id_cliente, cl.nombre, cl.apellido
HAVING COUNT(c.id_cita) > 5;
```

	nombre text	apellido text	total_visitas_trimestre bigint
1	MARIA ANGELES	MENDEZ	7

3. Servicios más rentables: calcular el total de ingresos por tipo de servicio y mostrar los 3 más rentables.

```
SELECT s.nombre AS servicio, SUM(CAST (c.precio AS decimal (10,2))) AS ingresos_totales
FROM servicios s
JOIN citas_servicios cs ON s.id_servicio = cs.id_servicio_servicios
JOIN citas c ON cs.id_cita_citas = c.id_cita
GROUP BY s.nombre
ORDER BY ingresos_totales DESC
LIMIT 3;
```

	servicio text	ingresos_totales numeric
1	Lavado + Corte	92.36
2	Tinte	90.93
3	Corte + Tinte	88.72

4. Promedio de duración por peluquera: obtener la duración media de todos los servicios realizados por cada peluquera.

```
SELECT p.nombre AS peluquera, AVG(s.duracion) AS duracion_media_servicio
FROM peluqueras p
JOIN servicios s ON p.id_peluquera = s.id_peluquera_peluqueras
GROUP BY p.nombre
ORDER BY duracion_media_servicio DESC;
```

	peluquera text	duracion_media_servicio numeric
1	DRISS	17.5000000000000000
2	ALBERTO	15.5000000000000000
3	IRENE	11.0000000000000000
4	JULIA	10.5000000000000000
5	ALVARO	10.0000000000000000
6	ANA MARIA	7.0000000000000000

5. Stock crítico: mostrar los productos con cantidad inferior al 10% de su stock máximo.

```
SELECT producto, stock_actual, stock_max
FROM inventario
WHERE stock_actual < (stock_max * 0.10);
```

	producto text	stock_actual integer	stock_max integer
1	Cera capilar	0	25
2	Pomada modelad...	2	23
3	Champú matizador	0	23
4	Spray de brillo	1	21
5	Tinte para el cabel...	0	25

6. Clientes VIP y gasto medio: mostrar el gasto medio total de los clientes VIP.

```
SELECT AVG(c.precio) AS gasto_medio_vip
FROM clientes cl
JOIN citas c ON cl.id_cliente = c.id_cliente_clientes
WHERE cl.vip = TRUE;
```

	gasto_medio_vip double precision
1	10.94916658848188

7. Ingresos por día: calcular el total de ingresos de la peluquería agrupados por día de la semana.

```
SELECT
    CASE EXTRACT(DOW FROM c.fecha)
        WHEN 0 THEN 'Domingo'
        WHEN 1 THEN 'Lunes'
        WHEN 2 THEN 'Martes'
        WHEN 3 THEN 'Miércoles'
        WHEN 4 THEN 'Jueves'
        WHEN 5 THEN 'Viernes'
        WHEN 6 THEN 'Sábado'
    END AS dia_semana,
    SUM(CAST (c.precio AS decimal (10,2))) AS ingresos_diarios
FROM citas c
GROUP BY dia_semana, EXTRACT(DOW FROM c.fecha)
ORDER BY EXTRACT(DOW FROM c.fecha);
```

	dia_semana text	ingresos_diarios numeric
1	Domingo	4009.91
2	Lunes	4414.83
3	Martes	4629.71
4	Miércoles	4434.54
5	Jueves	4636.54
6	Viernes	4447.00
7	Sábado	4471.51

8. Servicios combinados: mostrar a los clientes que se han realizado “corte” y “tinte” en la misma visita.

```
SELECT DISTINCT cl.nombre, cl.apellido
FROM clientes cl
JOIN citas c ON cl.id_cliente = c.id_cliente_clientes
JOIN citas_servicios cs ON c.id_cita = cs.id_cita_citas
JOIN servicios s ON cs.id_servicio_servicios = s.id_servicio
WHERE s.id_servicio = 7
ORDER BY cl.apellido, cl.nombre;
```

	nombre text	apellido text
1	ANGEL	CABRERA
2	ROBERTO	CABRERA
3	JORGE	FERRER
4	ANTONIA	MENDEZ
5	MARIA ANGELES	MENDEZ
6	JAVIER	MOLINA
7	CARLOS	SILVA
8	SONIA	VAZQUEZ

9. Ranking de productividad: crear una vista que muestre el número de servicios realizados por cada peluquera, su total facturado y su posición en el ranking.

```
CREATE VIEW ranking_productividad AS
SELECT p.nombre AS peluquera, COUNT(s.id_servicio) AS total_servicios_realizados,
SUM(CAST (c.precio AS decimal (10,2))) AS total_facturado,
RANK() OVER (
    ORDER BY COUNT(s.id_servicio) DESC, SUM(CAST (c.precio AS decimal (10,2))) DESC )
AS posicion_ranking
FROM peluqueras p
JOIN servicios s ON p.id_peluquera = s.id_peluquera_peluqueras
JOIN citas_servicios cs ON s.id_servicio = cs.id_servicio_servicios
JOIN citas c ON cs.id_cita_citas = c.id_cita
GROUP BY p.id_peluquera, p.nombre
ORDER BY posicion_ranking
LIMIT 5;

-- Consulta para ver el ranking
SELECT * FROM ranking_productividad;
```

	peluquera text	total_servicios_realizados bigint	total_facturado numeric	posicion_ranking bigint
1	JULIA	14	163.46	1
2	ALVARO	13	178.39	2
3	DRISS	11	120.74	3
4	ALBERTO	9	98.18	4
5	IRENE	5	60.56	5

10. Duración total de la jornada: sumar el tiempo total de cada peluquera que ha pasado atendiendo clientes durante una jornada completa.

```
SELECT p.nombre AS peluquera, SUM(s.duracion) AS duracion_total_servicio_minutos
FROM peluqueras p
JOIN servicios s ON p.id_peluquera = s.id_peluquera_peluqueras
JOIN citas_servicios cs ON s.id_servicio = cs.id_servicio_servicios
JOIN citas c ON cs.id_cita_citas = c.id_cita
WHERE c.fecha = DATE '2025-11-23'
GROUP BY p.nombre
ORDER BY duracion_total_servicio_minutos DESC;
```

	peluquera text	duracion_total_servicio_minutos bigint
1	ALBERTO	16

11. Clientes inactivos: listar los clientes que no han acudido en los últimos 6 meses.

```
SELECT cl.nombre, cl.apellido
FROM clientes cl
WHERE cl.id_cliente NOT IN (
    SELECT DISTINCT c.id_cliente_clientes
    FROM citas c
    WHERE c.fecha >= DATE '2025-11-18' - INTERVAL '6 months');
```

	nombre text	apellido text
1	RAFAEL	JIMENEZ
2	BEATRIZ	RIVERA
3	PILAR	GUTIERREZ
4	ALBERTO	SANTOS
5	MONICA	GARCIA
6	VICTOR	GIL
7	DIEGO	CANO
8	DOLORES	HERRERA
9	JOSE ANTONIO	VELASCO
10	MONTSERRAT	JIMENEZ
11	MANUEL	GARCIA
12	MARIO	MOLINA
13	DAVID	BRAVO
14	LUIS	CORTES
15	CARLOS	CARMONA
16	JOSE ANTONIO	GONZALEZ
17	VICTOR	CASTILLO
18	SONIA	HERNANDEZ
19	VICENTE	HERNANDEZ
20	ENRIQUE	DURAN
21	JUANA	FLORES
22	ENRIQUE	MORA
23	JUAN	SANCHEZ
24	JOSE ANTONIO	CRESPO

12. Descuento automático: simular una consulta que calcule un 10% de descuento a los clientes que hayan superado los 100€ en servicios durante el mes.

```
WITH GastoMensual AS (
    SELECT cl.id_cliente, cl.nombre, cl.apellido, SUM(CAST (c.precio AS decimal (10,2))) AS gasto_total_mes
    FROM clientes cl
    JOIN citas c ON cl.id_cliente = c.id_cliente_clientes
    WHERE c.fecha >= DATE '2025-11-18' - INTERVAL '30 days'
    GROUP BY cl.id_cliente, cl.nombre, cl.apellido)
SELECT nombre, apellido, gasto_total_mes, (gasto_total_mes * 0.10) AS descuento_sugerido
FROM GastoMensual
WHERE gasto_total_mes > 100
ORDER BY descuento_sugerido DESC;
```

	nombre text	apellido text	gasto_total_mes numeric	descuento_sugerido numeric

En este caso no aparecen datos pues no se cumple la condición, pero cambiando el intervalo de tiempo si aparecen datos.

```
WITH GastoMensual AS (
    SELECT cl.id_cliente, cl.nombre, cl.apellido, SUM(CAST (c.precio AS decimal (10,2))) AS gasto_total_mes
    FROM clientes cl
    JOIN citas c ON cl.id_cliente = c.id_cliente_clientes
    WHERE c.fecha >= DATE '2025-11-18' - INTERVAL '150 days'
    GROUP BY cl.id_cliente, cl.nombre, cl.apellido)
SELECT nombre, apellido, gasto_total_mes, (gasto_total_mes * 0.10) AS descuento_sugerido
FROM GastoMensual
WHERE gasto_total_mes > 100
ORDER BY descuento_sugerido DESC;
```

	nombre text	apellido text	gasto_total_mes numeric	descuento_sugerido numeric
1	ROBERTO	SANTOS	118.82	11.8820
2	ANGEL	VARGAS	110.95	11.0950
3	ALBERTO	JIMENEZ	104.31	10.4310

13. Uso de productos: calcular cuántas unidades de cada producto se han utilizado en los últimos 30 días (suponiendo relación entre servicios y productos).

```
SELECT i.producto, COUNT(c.id_cita) AS unidades_utilizadas
FROM inventario i
JOIN inventario_servicios ins ON i.id_producto = ins.id_producto_inventario
JOIN citas_servicios cs ON ins.id_servicio_servicios = cs.id_servicio_servicios
JOIN citas c ON cs.id_cita_citas = c.id_cita
WHERE c.fecha >= DATE '2025-11-23' - INTERVAL '30 days'
GROUP BY i.producto
ORDER BY unidades_utilizadas DESC;
```

	producto text	unidades_utilizadas bigint
1	Tinte para el cabello	4
2	Decolorante	2
3	Laca de fijación fuerte	2
4	Mascarilla capilar reparado...	1
5	Sérum capilar	1

14. Ingresos por método de pago: agrupar las facturas según el método de pago (efectivo, tarjeta, transferencia).

```
SELECT metodo_pago, SUM(importe) AS total_ingresos
FROM facturas
GROUP BY metodo_pago
ORDER BY total_ingresos DESC;
```

	metodo_pago text	total_ingresos double precision
1	tarjeta	15727.656657592126
2	efectivo	10362.903296311866
3	tarjeta de compra del Corte Inglés	5180.3179963528755

15. Servicios más largos: mostrar los 5 servicios cuya duración media haya sido mayor, junto con el nombre de las peluqueras que más los realizan.

```
SELECT s.nombre AS servicio, s.duracion AS duracion_media_minutos, p.nombre AS peluquera_asignada
FROM servicios s
JOIN peluqueras p ON s.id_peluquera_peluqueras = p.id_peluquera
ORDER BY duracion_media_minutos DESC
LIMIT 5;
```

	servicio text	duracion_media_minutos integer	peluquera_asignada text
1	Completo	25	DRISS
2	Corte + Tinte	16	ALBERTO
3	Corte + Peinado	15	ALBERTO
4	Lavado + Corte	13	ALVARO
5	Corte + Tinte + Peinado	12	JULIA

Transacción

```
BEGIN;
DO $$
DECLARE
    -- Variables para nuevos IDs
    nuevo_id_cliente INTEGER;
    nuevo_id_cita INTEGER;
    nuevo_id_factura INTEGER;

    -- Variables de la atención
    servicio_id CONSTANT INTEGER := 3;     -- Servicio: Tinte
    producto_id CONSTANT INTEGER := 1;     -- Producto: Tinte para el cabello
    nombre_cliente CONSTANT TEXT := 'JOSE';
    apellido_cliente CONSTANT TEXT := 'FERRER';

    -- Variables de control y stock
    precio_servicio DOUBLE PRECISION;
    stock_actual_producto INTEGER;

BEGIN

        -- Determinar los IDs disponibles buscando el maximo y sumandole 1 e introduciendolo en las nuevas variables
      SELECT COALESCE(MAX(id_cliente), 0) + 1 INTO nuevo_id_cliente FROM clientes;
    SELECT COALESCE(MAX(id_cita), 0) + 1 INTO nuevo_id_cita FROM citas;
      SELECT COALESCE(MAX(id_factura), 0) + 1 INTO nuevo_id_factura FROM facturas;

    -- Obtener precio del servicio
      SELECT precio INTO precio_servicio FROM servicios WHERE id_servicio = servicio_id;
```

```
    -- Bloquear la fila del producto
    SELECT stock_actual INTO stock_actual_producto
    FROM inventario
    WHERE id_producto = producto_id
    FOR UPDATE;

    -- Verifica el Stock
    IF stock_actual_producto < 1 THEN
        -- Si no hay stock, se lanza una excepción que forzará el ROLLBACK
        RAISE EXCEPTION 'STOCK INSUFICIENTE';
    END IF;

    -- Damos de Alta al Nuevo Cliente
    INSERT INTO clientes (id_cliente, nombre, apellido, vip, fecha_alta)
            VALUES (nuevo_id_cliente, nombre_cliente, apellido_cliente, FALSE,
CURRENT_DATE);

    -- Registramos la Factura
    INSERT INTO facturas (id_factura, importe, metodo_pago)
    VALUES (nuevo_id_factura, precio_servicio, 'efectivo');

    -- Registramos la Cita
            INSERT INTO citas (id_cita, fecha, precio, id_factura_facturas,
id_cliente_clientes)
      VALUES (nuevo_id_cita, CURRENT_DATE, precio_servicio, nuevo_id_factura,
nuevo_id_cliente);

    -- Asociar el Servicio a la Cita
    INSERT INTO citas_servicios (id_cita_citas, id_servicio_servicios)
    VALUES (nuevo_id_cita, servicio_id);

    -- Descontar Stock
    UPDATE inventario
    SET stock_actual = stock_actual - 1
```

```
    WHERE id_producto = producto_id;

-- Si pasa la excepcion anterior salta el rollback
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        -- Notificación final de que la transacción ha sido revertida.
            RAISE EXCEPTION 'TRANSACCIÓN FALLIDA: Se ha ejecutado el ROLLBACK.';
END $$;

-- Si el bloque DO se ejecutó sin excepciones, se aplica el COMMIT.
COMMIT;
```

Extras

Para este proyecto también se ofrecen ciertos extras para subir nota, de los cuales se han realizado:

1. Registrar automáticamente en la base de datos 500 clientes.

Se ha utilizado el programa de generación de datos para crear un script en ".txt" que genere 500 clientes completando todos sus atributos.

	id_cliente [PK] integer	nombre text	apellido text	vip boolean	fecha_alta date
1	1	ALVARO	NIETO	false	2025-11-24
2	2	JUAN MANUEL	MENDOZA	false	2025-03-03
3	3	ELENA	GOMEZ	false	2025-08-23
4	4	JOSE	ORTIZ	false	2025-02-01
5	5	RAFAEL	JIMENEZ	false	2025-11-10
6	6	RAFAEL	VEGA	false	2025-03-12
7	7	PEDRO	ALONSO	false	2025-01-26

Total rows: 500 | Query complete 00:00:00.088

Y este sería el código con el que se han generado los 500 clientes:

```
System.out.println("Empiezo con la tabla Clientes");
    try {
        String nombres[] = new String[100];
        String apellidos[] = new String[100];
        String rutaN = "C:\\Users\\Carlos\\Desktop\\Nombres.txt";
            String rutaA = "C:\\Users\\Carlos\\Desktop\\Apellidos.txt";
            FileReader fr1 = new FileReader(rutaN);
            BufferedReader br1 = new BufferedReader(fr1);
            FileReader fr2 = new FileReader(rutaA);
            BufferedReader br2 = new BufferedReader(fr2);
            for (int i=0;i<100;i++){
                nombres[i]=br1.readLine();
                //System.out.println(nombres[i]);
                apellidos[i]=br2.readLine();
                //System.out.println(apellidos[i]);
            }

                                        BufferedWriter bw = new BufferedWriter(new
FileWriter("C:\\Users\\Carlos\\Desktop\\Clientes_Tabla.txt"));

            // Se generan los 1000 datos
            for (int i = 1; i <= 500; i++) {
                int numero = (int)(Math.random() * 100);
                int numero2 = (int)(Math.random() * 100);
                  String fecha = ((int) (Math.random() * 28) + 1) + "/" + ((int) (Math.random()
* 12) + 1) + "/" + 2025;
                int visitas = (int) (Math.random() * 10) + 1;
                int vip;
            if (i % 50 == 0) {
                vip = 1;
            } else {
                vip = 0;
            }
```

```
            bw.write(i + "," + nombres[numero] + "," + apellidos[numero2] + "," + visitas +
"," + vip + "," + fecha);
            bw.newLine();
        }

            bw.close();
            System.out.println("Listo bb");
        } catch (IOException e) {
        }
```

5. Vistas dinámicas del estado actual de la peluquería

Se han creado diferentes vistas que son utilizadas en la interfaz para agilizar las consultas y no escribir tanto código sql en NetBeans:

```
/*Ranking de clientes con más visitas*/
CREATE VIEW clientesVisitas AS
SELECT cl.nombre, cl.apellido,COUNT(c.id_cita) AS "Visitas totales"
FROM clientes cl
JOIN citas c ON cl.id_cliente = c.id_cliente_clientes
GROUP BY cl.id_cliente, cl.nombre, cl.apellido
ORDER BY "Visitas totales" DESC
LIMIT 10;

/*Ranking de servicios más solicitados*/
CREATE VIEW serviciosSolicitados AS
SELECT s.nombre, COUNT(cs.id_cita_citas) AS "Solicitudes totales"
FROM servicios s
JOIN citas_servicios cs ON s.id_servicio = cs.id_servicio_servicios
GROUP BY s.nombre
ORDER BY "Solicitudes totales" DESC
LIMIT 5;

/*Ranking de dias con más citas*/
CREATE VIEW diasCitas AS
```

```
SELECT c.fecha, COUNT(c.id_cita) AS "Citas totales"
FROM citas c
GROUP BY c.fecha
ORDER BY "Citas totales" DESC
LIMIT 10;

/*Ranking de citas más caras*/
CREATE VIEW citasCaras AS
SELECT c.id_cita, CAST(c.precio as decimal (10,2)), cl.nombre, cl.apellido
FROM citas c
JOIN clientes cl ON c.id_cliente_clientes = cl.id_cliente
ORDER BY c.precio DESC
LIMIT 5;

/*Clientes que han pagado con tarjeta*/
CREATE VIEW clientesTarjeta AS
SELECT distinct cl.nombre, cl.apellido, f.metodo_pago
FROM clientes cl
JOIN citas c ON c.id_cliente_clientes = cl.id_cliente
JOIN facturas f ON c.id_factura_facturas = f.id_factura
WHERE f.metodo_pago='tarjeta';

/*Clientes que han pagado con efectivo*/
CREATE VIEW clientesEfectivo AS
SELECT distinct cl.nombre, cl.apellido, f.metodo_pago
FROM clientes cl
JOIN citas c ON c.id_cliente_clientes = cl.id_cliente
JOIN facturas f ON c.id_factura_facturas = f.id_factura
WHERE f.metodo_pago='efectivo';

/*Clientes que han pagado con tarjeta de compra del corte ingles*/
CREATE VIEW clientesCorteIngles AS
SELECT distinct cl.nombre, cl.apellido, f.metodo_pago
FROM clientes cl
```

JOIN citas c ON c.id_cliente_clientes = cl.id_cliente

JOIN facturas f ON c.id_factura_facturas = f.id_factura

WHERE f.metodo_pago='tarjeta de compra del Corte Inglés';

6. Conexión JDBC desde una aplicación Java para ejecutar consultas.

Para conectar la base de datos a la aplicación Java se ha utilizado un método de conexión en Apache NetBeans. Este crea una conexión con la clase “Connection” importada con la librería “*java.sql.Connection*”, además de las librerías “*java.sql.DriverManager*”, “*java.sql.ResultSet*” y “*java.sql.Statement*”, estas últimas se utilizan para conectar las consultas desde NetBeans a pgAdmin.
Este sería el código completo para realizar una consulta:

```
//conexion --> devuelve array clientes
public List<Object[]> Clientes() throws SQLException {
    //crea el String url donde pone la direccion de la base de datos
    String url = "jdbc:postgresql://localhost:5432/peluqueria";
    //se conecta con la base de datos mediante la libreria DriverManager pasando por parametro la url el usuario y la contraseña
    connection = DriverManager.getConnection(url, "postgres", "3434");
    //Crea el ArrayList de objetos datosClientes
    List<Object[]> datosClientes = new ArrayList<>();
    //Guarda en el String query la consulta a realizar
    String query = "SELECT id_cliente, nombre, apellido, vip, fecha_alta FROM clientes ORDER BY id_cliente";
    //Crea el el objeto statement que se usa en el objeto resulset para leer las lineas de la consulta
    try (Statement stmt = connection.createStatement(); ResultSet rs = stmt.executeQuery(query)) {
        //bucle para obtener los datos de la consulta
        while (rs.next()) {
            //Crea un arrays de objetos llamado fila
            Object[] fila = new Object[5];
            //Cada objeto del arrays se llena con los atributos de las tablas
            fila[0] = rs.getString("id_cliente");
            fila[1] = rs.getString("nombre");
            fila[2] = rs.getString("apellido");
            fila[3] = rs.getBoolean("vip");
            fila[4] = rs.getString("fecha_alta");
            //añade estos datos en orden al arrays datosClientes
            datosClientes.add(fila);
        }//poner catch de excepcion con popup por si falla
    }
    //devuelve el arrays datosClientes
    return datosClientes;
}
```

Luego para mostrar la tabla se usaría esta función en la clase deseada y se crearía el siguiente código.

```java
//Crea el método datosClientes
public void datosClientes() throws SQLException {
    //Crea un arrays con los nombres de las columnas de la tabla
    String[] nombresColumnas = {"ID Cliente", "Nombre", "Apellido", "VIP", "Fecha de alta"};
    //Copia el arrays que devuelve la funcion clientes de la clase cx en un arrays
    List<Object[]> datosClientes = cx.Clientes();
    //Crea un arrays multidimensional con las filas del array copiado y las columnas dadas en nombresColumnas
    Object[][] datosArray = new Object[datosClientes.size()][nombresColumnas.length];
    //Bucle para rellenar el arrays datosArray
    for (int i = 0; i < datosClientes.size(); i++) {
        datosArray[i] = datosClientes.get(i);
    }
    //Crea el modelo de tabla cogiendo los datos de los arrays
    DefaultTableModel modelo = new DefaultTableModel(datosArray, nombresColumnas);
    //Hace que la tablaClientes tenga el modelo propuesto antes
    tablaClientes.setModel(modelo);
    //Hace que la tabla haga un autoresize dependiendo del tamaño del arrays
    tablaClientes.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_ALL_COLUMNS);
}
```

7. Exportación de datos a CSV

Para la exportación de datos a csv se utilizará un código con lenguaje sql en la aplicación pgAdmin, en la “Query Tool”. Este sería el siguiente:

```sql
COPY public.clientes TO 'C:/Users/Carlos/Desktop/datos_csv/clientes.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');

COPY public.citas TO 'C:/Users/Carlos/Desktop/datos_csv/citas.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');

COPY public.facturas TO 'C:/Users/Carlos/Desktop/datos_csv/facturas.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');

COPY public.inventario TO 'C:/Users/Carlos/Desktop/datos_csv/inventario.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');

COPY public.peluqueras TO 'C:/Users/Carlos/Desktop/datos_csv/peluqueras.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');

COPY public.servicios TO 'C:/Users/Carlos/Desktop/datos_csv/servicios.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');

COPY public.citas_servicios TO 'C:/Users/Carlos/Desktop/datos_csv/citas_servicios.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');

COPY public.inventario_servicios TO 'C:/Users/Carlos/Desktop/datos_csv/inventario_servicios.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', ENCODING 'UTF8');
```

Esto crea una copia en la ruta señalada con formato CSV, esto se determina utilizando el comando “WITH (FORMAT CSV…)”, además se declara “HEADER” como false para que el archivo no tenga una cabecera indicando los nombres de las

columnas. El comando “DELIMITER” hace que el delimitador entre dato y dato sea una coma, y el comando “ENCODING UTF8” se utiliza para que el texto permita la letra “ñ” y las tildes.

8. Roles y permisos SQL diferenciados.

Para este apartado se han creado tres diferentes roles con permisos diferenciados. Estos roles son:

1. Administrador: Éste tiene permisos completos sobre toda la base de datos, pudiendo crear o eliminar nuevas tablas, columnas etc.
2. Gerente: Este simula el gerente de la peluquería, con permisos para alterar los datos de las ya creadas, pero sin poder crear ni eliminar nuevas tablas, solo los datos que hay en ellas.
3. Empleado: Este simula el empleado de la peluquería, pudiendo consultar los datos de la base de datos, pero sin poder alternarlos, ya sea creando cambiando o eliminando estos.

Para la creación de estos roles se ha utilizado el siguiente código:

```
/*Creación de rol administrador*/
CREATE ROLE administrador WITH LOGIN PASSWORD '1234';

/*Creación de rol de gerente*/
CREATE ROLE gerente WITH LOGIN PASSWORD '7891';

/*Creación de rol empleado*/
CREATE ROLE empleado WITH LOGIN PASSWORD '5678';

/*Otorgar permisos para administrador*/
GRANT ALL PRIVILEGES ON DATABASE peluqueria TO administrador;

/*Otorgar permisos para gerente*/
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO gerente;

/*Otorgar permisos para empleado*/
GRANT SELECT ON ALL TABLES IN SCHEMA public TO empleado;
```