PROYECTO FINAL BASES DE DATOS AVANZADAS

1. DEFINICIÓN DEL PROBLEMA

A. DESCRIPCIÓN DE FUNCIONALIDADES (APLICACIÓN)

Funcionalidad	Descripción	Usuario
Login	El login contará con 3 tipos de usuario. Dependiendo de con cual se ingrese, se tendrá acceso a diferentes funciones	admin cliente conductor
Pedir servicio	Función en la cual un usuario puede pedir un servicio dependiendo del tipo de servicio que necesite, la categoría con la que quiera el servicio, además otras opciones como ingresar la dirección hacia donde se debe dirigir el servicio.	cliente
Consultar servicio	En esta función los usuarios pueden ingresar a revisar los servicios que tengan y el estado en el que se encuentran.	admin cliente conductor
Terminar servicio	Función donde el usuario puede dar por terminado el servicio una vez haya sido entregado (función que será implementada dentro de Consultar servicio, pero solo podrá ser usada por el conductor).	conductor
Valorar Servicio	Función donde el usuario puede dar una valoración al servicio una vez terminado (función que será implementada dentro de Consultar servicio, pero solo podrá ser usada por el cliente)	cliente
Registrar usuarios	Función en la cual se permitirá registrar los usuarios por medio de un administrador	admin
Consultar datos sobre servicios	Función en la cual se podrá consultar los datos como el valor total de los servicios realizados, ya sea mensual o anualmente, los clientes que pidieron estos servicios, entre otras cosas.	admin

B. DESCRIPCIÓN DEL ENTORNO DE EJECUCIÓN (S.O., DBMS…..)

NetBeans

NetBeans es un entorno de desarrollo integrado de código abierto que permite crear aplicaciones de software en varios lenguajes de programación y proporciona herramientas para facilitar el proceso de desarrollo.

Elegimos este entorno de desarrollo teniendo en cuenta las siguientes ventajas del mismo:

Amplio soporte de lenguajes: Es compatible con múltiples lenguajes de programación, lo que brinda la versatilidad de trabajar con el lenguaje que mejor se adapte a las necesidades.

Integración con herramientas de desarrollo de Java: Está especialmente diseñado para el desarrollo de aplicaciones Java y ofrece una gran variedad de herramientas y características específicas para trabajar con este lenguaje de programación.

Compatible con múltiples plataformas: Es compatible con múltiples sistemas operativos, lo que brinda a los desarrolladores la capacidad de trabajar en diferentes plataformas sin tener que cambiar de herramienta de desarrollo.

PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto, que ofrece confiabilidad y escalabilidad. Soporta varios tipos de datos y ofrece características avanzadas.

Elegimos este motor de BD teniendo en cuenta las siguientes ventajas del mismo:

Soporte de múltiples lenguajes: Admite varios lenguajes de programación, lo que permite a los desarrolladores utilizar el lenguaje que mejor se adapte a sus necesidades.

Flexibilidad en el diseño de bases de datos: Ofrece flexibilidad en el diseño de bases de datos, lo que permite a los desarrolladores adaptar la estructura de la base de datos a sus necesidades específicas.

Alto rendimiento: PostgreSQL es una buena opción para aplicaciones que requieren procesamiento y almacenamiento de grandes volúmenes de datos, ya que está optimizado para un alto rendimiento en entornos de alta concurrencia.

C. DEFINICIÓN DE LAS REGLAS DE NEGOCIO (DELIMITAR PROBLEMA)

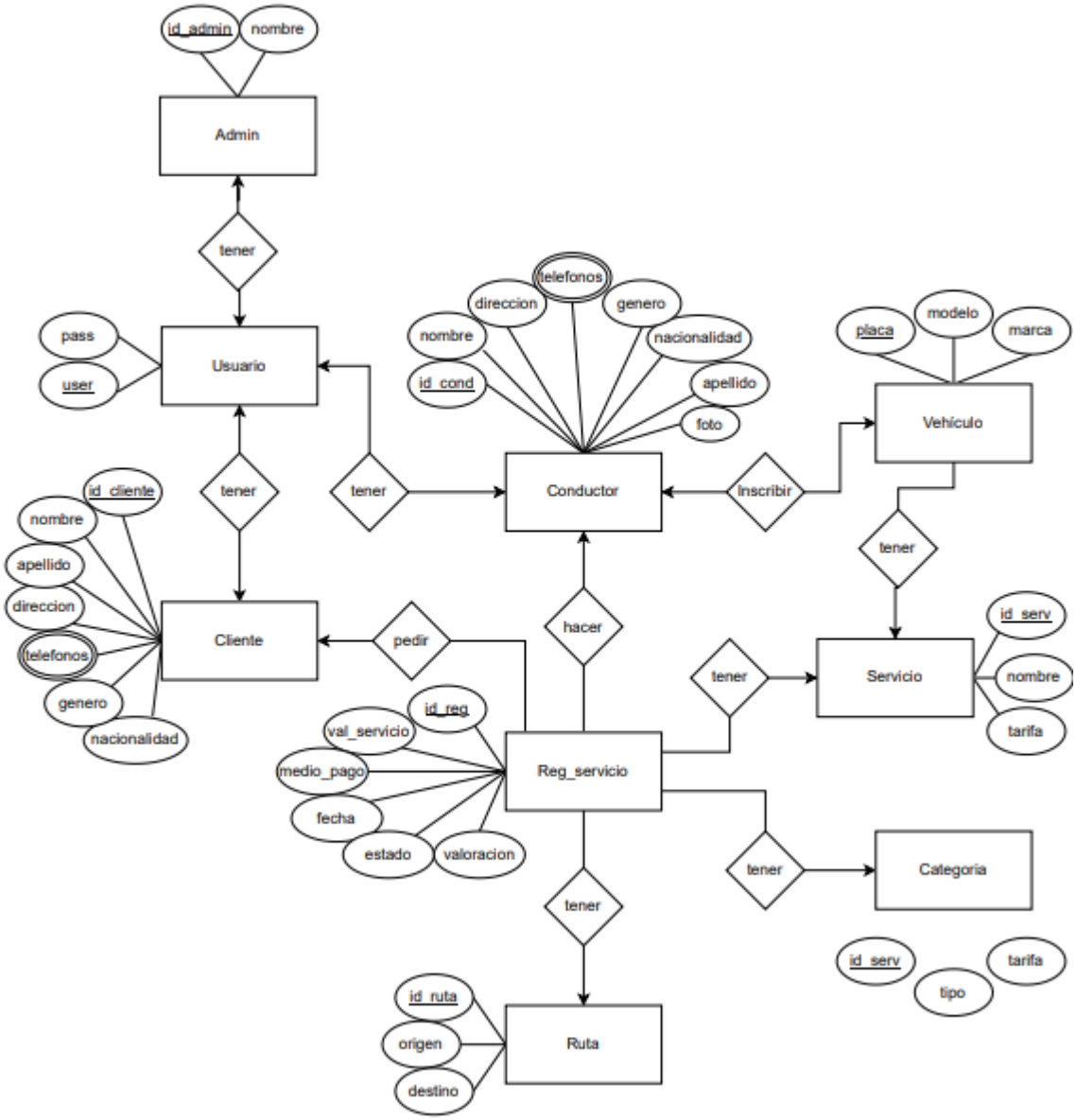
Reglas definidas en el enunciado

- Es necesario registrar los datos de los clientes contando con su identificación, nombre, dirección, teléfonos, género y nacionalidad.
- Los conductores inscriben su vehículo y la empresa requiere saber datos importantes como la identificación, nombre, dirección, teléfonos, género, nacionalidad y fotografía de los conductores.
- En cuanto a los vehículos se requiere la placa, el modelo, la marca y el tipo de servicio que prestará (pasajeros, alimentos o ambos).
- La compañía requiere que sea registrado cada servicio identificando el conductor; la ruta, que debe involucrar dirección de origen y dirección de destino; el cliente; el tipo de servicio y el valor del servicio.
- Cada servicio tiene una categoría que puede ser normal, especial o urgente y en cada caso la tarifa adicional puede variar, siendo la tarifa adicional más costosa la que corresponde a los servicios urgentes.

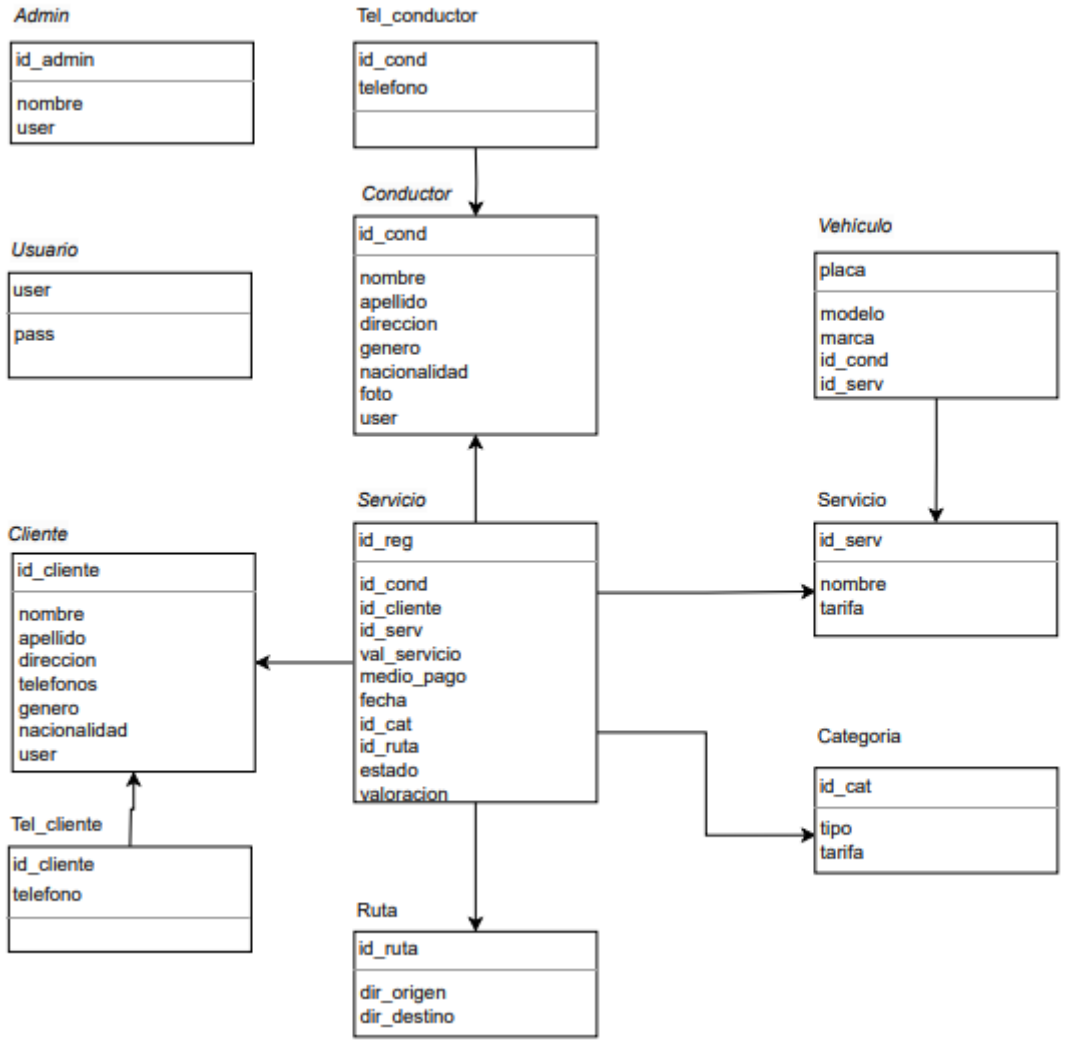
Reglas adicionales (Aplicación)

- Valores totales de los servicios realizados, diferenciando si corresponden a transporte de pasajeros o alimentos y la categoría a la que corresponde.
- Cantidad de servicios de pasajeros y de alimentos por mes.
- Datos de los clientes que han solicitado servicios durante un período de tiempo, organizados de mayor a menor.
- Valores totales de los servicios de acuerdo con su medio de pago.
- Habrá uno o más administradores, los cuales contarán con un usuario cada uno.
- El cliente podrá registrarse por sí mismo, o esperar que un administrador lo registre.
- El cliente podrá dar una valoración entre 1-5 una vez haya concluido el servicio solicitado.
- El conductor sólo podrá ser registrado por un administrador.

2. DIAGRAMA DEL MODELO ENTIDAD RELACIÓN



3. DIAGRAMA DEL MODELO RELACIONAL



4. DICCIONARIO DE DATOS

A. DESCRIPCIÓN DE TABLAS Y COLUMNAS

Tabla "conductor":

Columnas:

- id_cond (bigint, not null): identificador único del conductor.
- nombre (varchar(25), not null): nombre del conductor.
- apellido (varchar(25), not null): apellido del conductor.
- direccion (varchar(25)): dirección del conductor.
- genero (varchar(20)): género del conductor.
- nacionalidad (varchar(10)): nacionalidad del conductor.
- foto (oid): identificador de la imagen del conductor.
- usuario (varchar(10), not null): nombre de usuario del conductor.

Restricciones:

- primary key (id_cond): define la llave primaria de la tabla.
- foreign key (usuario) references usuario(usuario) on delete cascade: establece una llave foránea que referencia la columna "usuario" de la tabla "usuario" y realiza una eliminación en cascada.

Tabla "tel_conductor":

Columnas:

- id_cond (bigint, not null): identificador del conductor al que pertenece el número de teléfono.
- telefono (bigint, not null): número de teléfono del conductor.

Restricciones:

- primary key (id_cond, telefono): define una llave primaria compuesta por las columnas "id_cond" y "telefono".
- foreign key (id_cond) references conductor(id_cond) on delete cascade: establece una llave foranea que referencia la columna "id_cond" de la tabla "conductor" y realiza una eliminación en cascada.

Tabla "vehiculo":

Columnas:

- placa (varchar(10), not null): número de placa del vehículo.
- modelo (int, not null): modelo del vehículo.
- id_cond (bigint, not null): identificador del conductor asociado al vehículo.
- id_serv (serial, not null): identificador único del servicio.

Restricciones:

- primary key (placa): define la llave primaria de la tabla.
- foreign key (id_cond) references conductor(id_cond) on delete cascade: establece una llave foranea que referencia la columna "id_cond" de la tabla "conductor" y realiza una eliminación en cascada.
- foreign key (id_serv) references servicio(id_serv): establece una llave foranea que referencia la columna "id_serv" de la tabla "servicio".

Tabla "cliente":

Columnas:

- id_cliente (bigint, not null): identificador único del cliente.
- nombre (varchar(25), not null): nombre del cliente.
- apellido (varchar(25), not null): apellido del cliente.
- direccion (varchar(25), not null): dirección del cliente.
- genero (varchar(20)): género del cliente.
- nacionalidad (varchar(10)): nacionalidad del cliente.
- usuario (varchar(10), not null): nombre de usuario del cliente.

Restricciones:

- primary key (id_cliente): define la llave primaria de la tabla.
- foreign key (usuario) references usuario(usuario) on delete cascade: establece una llave foránea que referencia la columna "usuario" de la tabla "usuario" y realiza una eliminación en cascada.

Tabla "tel_cliente":

Columnas:

- id_cliente (bigint, not null): identificador del cliente al que pertenece el número de teléfono.
- telefono (bigint, not null): número de teléfono del cliente.

Restricciones:

- primary key (id_cliente, telefono): define una llave primaria compuesta por las columnas "id_cliente" y "telefono".
- foreign key (id_cliente) references cliente(id_cliente) on delete cascade: establece una llave foránea que referencia la columna "id_cliente" de la tabla "cliente" y realiza una eliminación en cascada.

Tabla "admin":

Columnas:

- id_admin (bigint, not null): identificador único del administrador.
- nombre (varchar(25), not null): nombre del administrador.
- usuario (varchar(10), not null): nombre de usuario del administrador.

Restricciones:

- primary key (id_admin): define la llave primaria de la tabla.
- foreign key (usuario) references usuario(usuario) on delete cascade: establece una llave foranea que referencia la columna "usuario" de la tabla "usuario" y realiza una eliminación en cascada.

Tabla "usuario":

Columnas:

- usuario (varchar(10), not null): nombre de usuario.
- pass (varchar(25), not null): contraseña del usuario.

Restricciones:

- primary key (usuario): define la llave primaria de la tabla.

Tabla "servicio":

Columnas:

- id_serv (serial, not null): identificador único del servicio.
- nombre (varchar(25), not null): nombre del servicio.
- tarifa (real): tarifa del servicio.

Restricciones:

- primary key (id_serv): define la clave primaria de la tabla.

Tabla "categoria":

Columnas:

- id_cat (serial, not null): identificador único de la categoría.
- nombre (varchar(20), not null): nombre de la categoría.
- tarifa (real, not null): tarifa de la categoría.

Restricciones:

- primary key (id_cat): define la clave primaria de la tabla.

Tabla "ruta":

Columnas:

- id_ruta (serial, not null): identificador único de la ruta.
- dir_origen (varchar(25), not null): dirección de origen de la ruta.
- dir_destino (varchar(25), not null): dirección de destino de la ruta.

Restricciones:

- primary key (id_ruta): define la clave primaria de la tabla.

Tabla "reg_servicio":

Columnas:

- id_reg (serial, not null): identificador único del registro de servicio.
- id_cond (bigint, not null): identificador del conductor asociado al servicio.
- id_cliente (bigint, not null): identificador del cliente asociado al servicio.
- id_serv (serial, not null): identificador del servicio.
- val_servicio (real, not null): valor del servicio.
- medio_pago (varchar(30), not null): método de pago utilizado.
- fecha (varchar(10), not null): fecha del servicio.
- id_cat (serial, not null): identificador de la categoría del servicio.
- id_ruta (serial, not null): identificador de la ruta del servicio.
- estado (varchar(15)): estado del servicio.
- valoracion (varchar(1)): valoración del servicio.

Restricciones:

- primary key (id_reg): define la clave primaria de la tabla.
- foreign key (id_cond) references conductor(id_cond): establece una clave externa que referencia la columna "id_cond" de la tabla "conductor".
- foreign key (id_cliente) references cliente(id_cliente) on delete cascade: establece una clave externa que referencia la columna "id_cliente" de la tabla "cliente" y realiza una eliminación en cascada.
- foreign key (id_serv) references servicio(id_serv): establece una clave externa que referencia la columna "id_serv" de la tabla "servicio".
- foreign key (id_cat) references categoria(id_cat): establece una clave externa que referencia la columna "id_cat" de la tabla "categoria".

- foreign key (id_ruta) references ruta(id_ruta): establece una clave externa que referencia la columna "id_ruta" de la tabla "ruta".

Tabla "log_reg_servicio":

Columnas:

- id_reg (serial): identificador del registro de servicio.
- id_cond (bigint): identificador del conductor asociado al servicio.
- id_cliente (bigint): identificador del cliente asociado al servicio.
- id_serv (serial): identificador del servicio.
- val_servicio (real): valor del servicio.
- medio_pago (varchar(30)): método de pago utilizado.
- fecha (varchar(10)): fecha del servicio.
- id_cat (serial): identificador de la categoría del servicio.
- id_ruta (serial): identificador de la ruta del servicio.
- estado (varchar(15)): estado del servicio.
- valoracion (varchar(1)): valoración del servicio.

B. DESCRIPCIÓN DE PROCEDIMIENTOS Y TRIGGERS

Procedimiento "calc_valorserv":

- Descripción: Este procedimiento recibe dos valores numéricos como parámetros y devuelve la suma de esos valores.
- Parámetros de entrada: x (real), y (real)
- Tipo de retorno: real
- Funcionamiento: El procedimiento realiza una simple suma de los parámetros de entrada.

```
create function  calc_valorserv(x real, y real)
returns real
as $$
    select x+y;
$$
language sql;
```

Procedimiento "cond_disponible":

- Descripción: Este procedimiento devuelve un identificador de conductor disponible para realizar un nuevo servicio. El conductor se selecciona aleatoriamente entre aquellos que no están ocupados actualmente y cuyo vehículo coincide con el tipo de servicio especificado.
- Parámetros de entrada: serv (entero)
- Tipo de retorno: long

- Funcionamiento: El procedimiento utiliza una consulta SQL para seleccionar un conductor aleatorio que cumpla con las condiciones especificadas.

```
create function cond_disponible(serv int)
returns long
as $$
   select c.id_cond from conductor c
          inner join vehiculo v using (id_cond)
          where (c.id_cond not in (select id_cond from reg_servicio)
                    and (v.id_serv=serv or v.id_serv=3))
          or (c.id_cond not in (select id_cond from reg_servicio
                    where estado = 'En camino')
                    and (id_serv=serv or id_serv=3))
          order by random() limit 1;
$$
language sql;
```

Procedimiento "valor_total_servicios":

- Descripción: Este procedimiento devuelve una tabla con el nombre del servicio, el nombre de la categoría y el valor total de los servicios realizados agrupados por tipo de servicio y categoría.
- Tipo de retorno: tabla (servicio nom_serv, categoria nom_cat, valor_total real)
- Funcionamiento: El procedimiento utiliza una consulta SQL para unir las tablas "reg_servicio", "servicio" y "categoria" y calcular la suma de los valores de servicio agrupados por servicio y categoría.

```
create function valor_total_servicios()
returns table(servicio nom_serv, categoria nom_cat, valor_total real)
as $$
    select s.nombre, c.nombre, sum(rs.val_servicio)
    from reg_servicio rs
    inner join servicio s using (id_serv)
    inner join categoria c using (id_cat)
    group by s.nombre, c.nombre;
$$
language sql;
```

Procedimiento "cant_servicios_mes":

- Descripción: Este procedimiento devuelve una tabla con el mes, la cantidad de servicios de pasajeros y la cantidad de servicios de alimentos realizados, agrupados por mes.
- Tipo de retorno: tabla (mes varchar, servicios_pasajeros bigint, servicios_alimentos bigint)
- Funcionamiento: El procedimiento utiliza una consulta SQL para calcular la cantidad de servicios de pasajeros y servicios de alimentos realizados agrupados por mes.

```
create function cant_servicios_mes()
returns table(mes varchar, servicios_pasajeros bigint, servicios_alimentos bigint)
as $$
    select to_char(fecha::date, 'tmmonth') as mes,
            count(case when id_serv=1 then 1 end),
            count(case when id_serv=2 then 1 end)
    from reg_servicio
    group by mes
    order by mes desc;
$$
language sql;
```

Procedimiento "servs_solicitados_xtiempo":

- Descripción: Este procedimiento devuelve una tabla con los datos de los clientes que han solicitado servicios en un período de tiempo especificado.
- Parámetros de entrada: fch1 (varchar), fch2 (varchar)
- Tipo de retorno: tabla (id_cliente bigint, nombre varchar, apellido varchar, direccion varchar, genero varchar, nacionalidad varchar, usuario varchar, fecha_pedido varchar)
- Funcionamiento: El procedimiento utiliza una consulta SQL para seleccionar los datos distintos de los clientes que han solicitado servicios dentro del rango de fechas especificado.

```
create function servs_solicitados_xtiempo(fch1 varchar, fch2 varchar)
returns table
(id_cliente bigint, nombre varchar, apellido varchar,
 direccion varchar, genero varchar, nacionalidad varchar,
 usuario varchar, fecha_pedido varchar)
```

```
as $$
    select distinct c.*, rs.fecha from cliente c
    inner join reg_servicio rs using (id_cliente)
    where fecha between fch1 and fch2
  order by c.id_cliente, rs.fecha desc;
$$
language sql;
```

Procedimiento "valor_servs_mediopago":

- Descripción: Este procedimiento devuelve una tabla con el medio de pago y el valor total de los servicios realizados agrupados por medio de pago.
- Tipo de retorno: tabla (medio_pago m_pago, valor_total real)
- Funcionamiento: El procedimiento utiliza una consulta SQL para calcular la suma de los valores de servicio agrupados por medio de pago.

```
create function valor_servs_mediopago()
returns table(medio_pago m_pago, valor_total real)
as $$
    select rs.medio_pago, sum(rs.val_servicio)
    from reg_servicio rs
    group by rs.medio_pago;
$$
language sql;
```

Trigger "log_servs_tr":

- Descripción: Este trigger se activa después de eliminar un registro de la tabla "reg_servicio" y guarda los datos del registro en la tabla "log_reg_servicio" antes de que se elimine el cliente asociado al servicio.
- Tipo de evento: DELETE en la tabla "reg_servicio"
- Cuerpo del trigger: El trigger ejecuta el procedimiento "log_servs" que realiza la inserción de los datos en la tabla "log_reg_servicio".

```
create function log_servs()
returns trigger
as $$
Begin
    insert into "log_reg_servicio" values
```

```
        (old.id_reg, old.id_cond, old.id_cliente, old.id_serv,
         old.val_servicio, old.medio_pago, old.fecha,
         old.id_cat, old.id_ruta, old.estado, old.valoracion);
        return new;
    End;
    $$
    language plpgsql;


    create trigger log_servs_tr
    after Delete on reg_servicio
    for each row
    execute function log_servs();
```

C. DESCRIPCIÓN DE VISTAS

Vista "servs_cliente_vw":

- Descripción: Esta vista permite al cliente visualizar solo los datos necesarios de sus pedidos realizados. Muestra el número de registro, la identificación del cliente, el nombre del cliente, el nombre del servicio, el nombre de la categoría, el valor del servicio, la fecha y el estado del servicio.
- Columnas de la vista:
  - NroRegistro: Número de registro del servicio.
  - identificacion: Identificación del cliente.
  - nombre_cliente: Nombre del cliente.
  - servicio: Nombre del servicio.
  - categoria: Nombre de la categoría del servicio.
  - valor_servicio: Valor del servicio.
  - fecha: Fecha del servicio.
  - estado: Estado del servicio.

```
create view servs_cliente_vw as

select rs.id_reg as NroRegistro, cl.id_cliente as identificacion, cl.nombre as
nombre_cliente, s.nombre as servicio, c.nombre as categoria, rs.val_servicio as
valor_servicio, rs.fecha, rs.estado

from reg_servicio rs

inner join cliente cl using (id_cliente)

inner join servicio s using (id_serv)

inner join categoria c using (id_cat);
```

Vista "servs_cond_vw":

- Descripción: Esta vista permite al conductor visualizar solo los datos necesarios de los pedidos que debe hacer o ha realizado. Muestra el número de registro, la identificación del conductor, el nombre del conductor, el nombre del servicio, el nombre de la categoría, el valor del servicio, la dirección de origen, la dirección de destino, la fecha y el estado del servicio.
- Columnas de la vista:
  - NroRegistro: Número de registro del servicio.
  - identificacion: Identificación del conductor.
  - nombre_condutor: Nombre del conductor.
  - servicio: Nombre del servicio
  - categoria: Nombre de la categoría del servicio.
  - valor_servicio: Valor del servicio.
  - origen: Dirección de origen del servicio.
  - destino: Dirección de destino del servicio.
  - fecha: Fecha del servicio.
  - estado: Estado del servicio

```
create view servs_cond_vw as

select rs.id_reg as NroRegistro, cl.id_cond as identificacion, cl.nombre as
nombre_conductor, s.nombre as servicio, c.nombre as categoria, rs.val_servicio as
valor_servicio, r.dir_origen as origen, r.dir_destino as destino, rs.fecha, rs.estado

from reg_servicio rs

inner join conductor cl using (id_cond)

inner join servicio s using (id_serv)

inner join categoria c using (id_cat)

inner join ruta r using (id_ruta);
```

Vista "servs_admin_vw":

- Descripción: Esta vista permite al administrador visualizar todos los datos de los pedidos. Muestra el número de registro, la identificación del conductor y del cliente, el nombre del servicio, el nombre de la categoría, el valor del servicio, la dirección de origen, la dirección de destino, la fecha, el estado del servicio y la valoración del servicio.
- Columnas de la vista:
  - NroRegistro: Número de registro del servicio.
  - conductor: Identificación del conductor.
  - Cliente: Identificación del cliente
  - servicio: Nombre del servicio
  - categoria: Nombre de la categoría del servicio.
  - valor_servicio: Valor del servicio.

- origen: Dirección de origen del servicio.
- destino: Dirección de destino del servicio.
- fecha: Fecha del servicio.
- estado: Estado del servicio.
- valoracion: Valoración del servicio.

```
create view servs_admin_vw as

select rs.id_reg as NroRegistro, rs.id_cond as conductor, rs.id_cliente as cliente,
s.nombre as servicio, c.nombre as categoria, rs.val_servicio as valor_servicio,
r.dir_origen as origen, r.dir_destino as destino, rs.fecha, rs.estado, rs.valoracion

from reg_servicio rs

inner join servicio s using (id_serv)

inner join categoria c using (id_cat)

inner join ruta r using (id_ruta)
```

5. CÓDIGO FUENTE:

A. ESTRUCTURA DE LA BASE DE DATOS:

TABLAS SIN RESTRICCIONES

```
create table conductor(
   id_cond bigint not null,
   nombre varchar(25) not null,
   apellido varchar(25) not null,
   direccion varchar(25),
   genero varchar(20),
   nacionalidad varchar(10),
   foto oid,
   usuario varchar(10) not null,
   primary key (id_cond)
);
create table tel_conductor(
   id_cond bigint not null,
   telefono bigint not null
);

create table vehiculo(
   placa varchar(10) not null,
   modelo int  not null,
   id_cond bigint not null,
   id_serv serial not null,
   primary key (placa)
);
create table cliente(
   id_cliente bigint not null,
   nombre varchar(25) not null,
   apellido varchar(25) not null,
   direccion varchar(25) not null,
   genero varchar(20),
   nacionalidad varchar(10),
   usuario varchar(10) not null,
   primary key (id_cliente)
);
create table tel_cliente(
   id_cliente bigint not null,
   telefono bigint not null
);
create table admin(
   id_admin bigint not null,
   nombre varchar(25) not null,
   usuario varchar(10) not null,
   primary key(id_admin)
);
create table usuario(
   usuario varchar(10) not null,
   pass varchar(25) not null,
   primary key(usuario)
);
create table servicio(
   id_serv serial not null,
   nombre varchar(25) not null,
   tarifa real,
   primary key(id_serv)
);
create table categoria(
   id_cat serial not null,
```

```
    nombre varchar(20) not null,
    tarifa real not null,
    primary key(id_cat)
);
create table ruta(
    id_ruta serial not null,
    dir_origen varchar(25) not null,
    dir_destino varchar(25) not null,
    primary key(id_ruta)
);
create table reg_servicio(
    id_reg serial not null,
    id_cond bigint not null,
    id_cliente bigint not null,
    id_serv serial not null,
    val_servicio real not null,
    medio_pago varchar(30) not null,
    fecha varchar(10) not null,
    id_cat serial not null,
    id_ruta serial not null,
    estado varchar(15),
    valoracion varchar(1),
    primary key(id_reg)
);
create table log_reg_servicio(
    id_reg serial,
    id_cond bigint,
    id_cliente bigint,
    id_serv serial,
    val_servicio real,
    medio_pago varchar(30),
    fecha varchar(10),
    id_cat serial,
    id_ruta serial,
    estado varchar(15),
    valoracion varchar(1)
);
```

RESTRICCIONES

```
create domain gen varchar
not null
check(value= 'Femenino' or value='Masculino'
or value='No Binario' or value='No
Especificado');

create type nom_serv as enum
('Pasajeros','Alimentos','Todos');
create type nom_cat as enum
('Normal','Especial','Urgente');
create type dir as enum ('CL 90 C SUR 1-45');
create type m_pago as enum
('Tarjeta','Efectivo');
create type est as enum ('En
camino','Terminado');
create type valoracion as enum ('1','2','3','4','5');

alter table conductor
add constraint conduser_fk
foreign key(usuario) references
usuario(usuario) on delete cascade,
alter column genero type gen;
alter table conductor
alter column genero set default 'No
Especificado';

alter table tel_conductor
add constraint tlfcond_pk
primary key(id_cond, telefono),
add constraint tlfcond_fk
foreign key(id_cond) references
conductor(id_cond) on delete cascade;

alter table vehiculo
add constraint vidcond_fk
foreign key(id_cond) references
conductor(id_cond) on delete cascade,
add constraint vidserv_fk
foreign key(id_serv) references
servicio(id_serv);

alter table cliente
add constraint cluser_fk
foreign key(usuario) references
usuario(usuario) on delete cascade,
alter column genero type gen;
alter table cliente
alter column genero set default 'No
Especificado';
```

```
alter table tel_cliente
add constraint tlfcl_pk
primary key(id_cliente, telefono),
add constraint tlfcl_fk
foreign key(id_cliente) references
cliente(id_cliente) on delete cascade;

alter table admin
add constraint admuser_fk
foreign key(usuario) references
usuario(usuario) on delete cascade;

alter table servicio
alter column nombre type nom_serv USING
nombre::nom_serv;

alter table categoria
alter column nombre type nom_cat USING
nombre::nom_cat;

alter table ruta
alter column  dir_origen type dir USING
dir_origen::dir,
alter column dir_origen set default 'CL 90 C
SUR 1-45';
```

```
alter table reg_servicio
add constraint rgcond_fk
foreign key(id_cond) references
conductor(id_cond);

alter table reg_servicio
add constraint rgcl_fk
foreign key(id_cliente) references
cliente(id_cliente) on delete cascade,
add constraint rgserv_fk
foreign key(id_serv) references
servicio(id_serv);

alter table reg_servicio
alter column medio_pago type m_pago USING
medio_pago::m_pago,
alter column estado type est USING
estado::est;

alter table reg_servicio
alter column valoracion type valoracion USING
valoracion::valoracion;
```

B. INSERCIÓN DE DATOS:

Para la inserción de datos a la BD, mostramos algunos de los métodos utilizados en la aplicación:

```
public String insertar(Cliente c){
String mensaje="";
    try {
        con=new Conexion();
        PreparedStatement consulta = null;
        con.conectar();
        String comando= "insert into cliente values(?,?,?,?,?,?,?)";
        consulta=con.getConexion().prepareStatement( sql:comando);
        consulta.setLong( parameterIndex: 1, x: c.getIdCliente());
        consulta.setString( parameterIndex: 2, x: c.getNombre());
        consulta.setString( parameterIndex: 3, x: c.getApellido());
        consulta.setString( parameterIndex: 4, x: c.getDireccion());
        consulta.setString( parameterIndex: 5, x: c.getGenero());
        consulta.setString( parameterIndex: 6, x: c.getNacionalidad());
        consulta.setString( parameterIndex: 7, x: c.getUsuario());
        consulta.execute();
        mensaje="Registro exitoso...";
        consulta.close();
        con.getConexion().close();
    } catch (SQLException ex) {
       mensaje="Error en metodo insertar de ClienteDAO...\n"+ex.getMessage();
    }
    return mensaje;
}
```

```java
public String insertar(Conductor c){
String mensaje="";
    try {
        con=new Conexion();
        PreparedStatement consulta = null;
        con.conectar();
        String comando= "insert into conductor values(?,?,?,?,?,?,?,?)";
        consulta=con.getConexion().prepareStatement( sql:comando);
        consulta.setLong( parameterIndex:1, x:c.getIdCond());
        consulta.setString( parameterIndex:2, x:c.getNombre());
        consulta.setString( parameterIndex:3, x:c.getApellido());
        consulta.setString( parameterIndex:4, x:c.getDireccion());
        consulta.setString( parameterIndex:5, x:c.getGenero());
        consulta.setString( parameterIndex:6, x:c.getNacionalidad());
        consulta.setInt( parameterIndex:7,  x:c.getFoto());
        consulta.setString( parameterIndex:8, x:c.getUsuario());
        consulta.execute();
        mensaje="Registro exitoso...";
        consulta.close();
        con.getConexion().close();
    } catch (SQLException ex) {
       mensaje="Error en metodo insertar de ConductorDAO...\n"+ex.getMessage();
    }
    return mensaje;
}
```

```java
    public String insertar(String dir){
        String mensaje="";
        try {
            con=new Conexion();
            con.conectar();
            PreparedStatement consulta = null;
            String comando= "insert into ruta (dir_destino) values(?)";
            consulta=con.getConexion().prepareStatement( sql:comando);
            consulta.setString( parameterIndex:1, x:dir);
            consulta.execute();
            mensaje="Registro exitoso...";
            consulta.close();
            con.getConexion().close();
        } catch (SQLException ex) {
           mensaje="Error en metodo insertar de RutaDAO...\n"+ex;
        }
        return mensaje;
    }
```

```java
    public String insertar(Usuario u){
        String mensaje="";
        try {
            con=new Conexion();
            con.conectar();
            PreparedStatement consulta = null;
            String comando= "insert into usuario values(?,?)";
            consulta=con.getConexion().prepareStatement( sql:comando);
            consulta.setString( parameterIndex:1, x:u.getUsuario());
            consulta.setString( parameterIndex:2, x:u.getPass());
            consulta.execute();
            mensaje="Registro exitoso...";
            consulta.close();
            con.getConexion().close();
        } catch (SQLException ex) {
           mensaje="Error en metodo insertar de UsuarioDAO...\n"+ex;
        }
        return mensaje;
    }
```

Los anteriores métodos pertenecen a los archivos DAO del proyecto. El objetivo principal de este código es insertar los datos en la base de datos. El método insertar realiza los siguientes pasos:

- Crea una instancia de la clase Conexión para establecer una conexión con la base de datos.
- Prepara una consulta SQL de inserción que incluye los datos del cliente.
- Establece los valores de los parámetros de la consulta utilizando los atributos del objeto.
- Ejecuta la consulta para insertar el nuevo registro en la base de datos.
- Cierra la consulta y la conexión con la base de datos.
- Devuelve un mensaje que indica si la operación de inserción fue exitosa o si se produjo un error.

C. CONEXIÓN DE LA APLICACIÓN A LA BD

La clase Conexion tiene varios atributos privados que representan los datos necesarios para establecer la conexión con la base de datos:

- conexion: Objeto de tipo Connection que representa la conexión establecida con la base de datos.
- bd: Nombre de la base de datos.
- usuario: Nombre de usuario para acceder a la base de datos.
- clave: Contraseña del usuario para acceder a la base de datos.
- mensaje: Mensaje de estado de la conexión.

La clase tiene dos constructores:

- El primer constructor recibe todos los atributos de conexión como parámetros y asigna los valores correspondientes.
- El segundo constructor no recibe parámetros y establece valores predeterminados para los atributos de conexión.

```java
package modelodao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Conexion {
  private Connection conexion;

  private String bd, usuario, clave, mensaje;

    public Conexion(Connection conexion, String bd, String usuario, String clave, String mensaje) {
        this.conexion = conexion;
        this.bd = bd;
        this.usuario = usuario;
        this.clave = clave;
        this.mensaje = mensaje;
    }

    public Conexion() {
        this.conexion = null;
        this.bd = "proyfinal";
        this.usuario = "postgres";
        this.clave = "admin";
        this.mensaje = "";
    }
```

El método conectar() es el encargado de establecer la conexión con la base de datos. Utiliza la clase DriverManager para cargar el controlador de PostgreSQL y establecer la conexión utilizando la URL de conexión, el nombre de usuario y la contraseña proporcionados. El resultado de la conexión se asigna al atributo conexion, y se establece el mensaje de estado de la conexión.

```
public void conectar(){
    try {
        Class.forName( className:"org.postgresql.Driver");
        String ruta="jdbc:postgresql://localhost:5432/"+bd;
        System.out.println( x:ruta);
        conexion= DriverManager.getConnection( url:ruta, user:usuario, password:clave);
        mensaje="Conexión exitosa...";
    } catch (ClassNotFoundException ex) {
        mensaje="No se pudo establecer conexion...";
    } catch (SQLException ex) {
        mensaje=" No se puede conectar con MySQL...";
    }
}
```

El método toString() devuelve una representación en forma de cadena de la instancia de la clase Conexion, que incluye los valores de todos los atributos.

```
@Override
public String toString() {
    return "Conexion{" + "conexion=" + conexion + ", bd=" + bd + ", usuario=" + usuario + ", clave=" + clave + ", mensaje=" + mensaje + '}';
}
```

La clase también proporciona getters y setters para acceder a los atributos.

```
public Connection getConexion() {
    return conexion;
}

public void setConexion(Connection conexion) {
    this.conexion = conexion;
}

public String getBd() {
    return bd;
}

public void setBd(String bd) {
    this.bd = bd;
}

public String getUsuario() {
    return usuario;
}

public void setUsuario(String usuario) {
    this.usuario = usuario;
}

public String getClave() {
    return clave;
}

public void setClave(String clave) {
    this.clave = clave;
}

public String getMensaje() {
    return mensaje;
}

public void setMensaje(String mensaje) {
    this.mensaje = mensaje;
}
```

6. DESCRIPCIÓN DE LA APLICACIÓN (MANUAL DE USUARIO)

Para describir el funcionamiento de la aplicación, lo separamos en 3 secciones

ADMIN

CLIENTE

CONDUCTOR