Alejandro Flechas - 2024121288
Camilo Romero - 202411065
Simon Pineda - 202410683

Análisis:

RF1 - REGISTRAR UNA CIUDAD

Registra una nueva ciudad en la que se prestarán servicios de transporte para **ALPESCAB**.

RF2 - REGISTRAR UN USUARIO DE SERVICIOS

Un usuario puede darse de alta en la aplicación, usando los datos descritos en la descripción general del caso.

RF3 - REGISTRAR UN USUARIO CONDUCTOR

Un usuario que va a prestar servicios de transporte puede darse de alta en la aplicación, usando los datos descritos.

RF4 - REGISTRAR UN VEHÍCULO PARA UN USUARIO CONDUCTOR

Un conductor puede registrar un vehículo a su nombre con los datos indicados en la descripción general del caso.

RF5 - REGISTRAR LA DISPONIBILIDAD DE UN USUARIO CONDUCTOR Y SU VEHÍCULO PARA UN SERVICIO.

Un conductor puede seleccionar días y rangos horarios en los cuales prestará diferentes tipos de servicios. (transporte de pasajeros, entrega de comida, transporte de mercancías. En caso de prestar el servicio de transporte de pasajeros, el modelo y capacidad del vehículo determinarán el nivel de este (Estándar, Confort y Large). No es posible tener disponibilidades que se superpongan para un mismo conductor.

RF6 - MODIFICAR LA DISPONIBILIDAD DE UN VEHÍCULO PARA SERVICIOS

Un conductor puede modificar la disponibilidad en términos de días y horarios para las que prestará un servicio con su vehículo. No debe ser posible cambiar la disponibilidad si esta se superpone con otras disponibilidades del mismo conductor.

RF7 - REGISTRAR UN PUNTO GEOGRÁFICO

Todos los usuarios pueden registrar puntos geográficos en cualquier momento. Esto requiere indicar un nombre (para establecimientos particulares), una dirección, unas coordenadas geográficas y una ciudad.

RF8 - SOLICITAR UN SERVICIO POR PARTE DE UN USUARIO DE SERVICIOS

Un usuario puede solicitar un servicio particular, indicando un punto de partida y un punto de llegada. La aplicación calcula la tarifa si existen diferentes niveles, realiza un cobro al usuario y automáticamente le asigna un conductor que tenga disponibilidad, que se encuentre cerca y que no esté ya asignado a un servicio en este momento.

RF9 - REGISTRAR UN VIAJE PARA UN USUARIO DE SERVICIOS Y UN USUARIO CONDUCTOR

Una vez un servicio ha sido prestado, el conductor se marca disponible nuevamente, y se registra en el histórico de conductor y pasajero los datos del viaje, incluyendo la hora de inicio y hora de finalización, la longitud, el costo y demás información relevante.

RF10 - DEJAR UNA REVISIÓN POR PARTE DEL USUARIO DE SERVICIOS

Un pasajero puede registrar una revisión a un conductor para un servicio prestado, dando una calificación y dejando un comentario de texto.

RF11 - DEJAR UNA REVISIÓN POR PARTE DE UN USUARIO CONDUCTOR

Un conductor puede registrar una revisión a un cliente para un servicio prestado, dando una calificación y dejando un comentario de texto.

RF1: Nuestro sistema registra de manera adecuada las ciudades en las que se emplea el servicio, y al depender de una llave única, estas no se pueden repetir generando errores.
Prueba Valida: (“Cartagena”) -> es valida ya que la primary key es sistem assigned y el nombre no es null
Prueba Invalida ()-> Al ser null no es valido

RF2: Como se muestra en el uml y en el sistema relacional, el usuario de servicio es capaz de borrar y crear su cuenta, sin repetidos en la información.
Prueba Valida: (“1001”,”1029144444”, “1029144444”)-> todas son llaves validas y ninguna puede ser Null
Prueba Inválida (”1029144444”, “1029144444”)-> se omite la primary key, por lo cual es válida

RF3: De la misma manera que la anterior, el usuario que es conductor puede crear y eliminar su usuario.
Prueba Válida: (“1001”,”1029144444”, “True”)-> todas son llaves válidas y ninguna puede ser Null
Prueba Inválida (”1029144444”, “1029144444”)-> se omite la primaery key, por lo cual es válida

RF4: El usuario que sea conductor es capaz de añadir un carro con los parámetros adecuados y sin repeticiones, por lo cual se cumpliría el requerimiento.
Prueba Válida: (“aaa111”,”Toyota”,”corola”,”premium”,”negro”,5)-> todas son llaves válidas y todas las NN están
Prueba Inválida (“aaa111”,”Toyota”,Null,”premium”,”negro”,5)->No es valido porque no tiene modelo

RF5: Utilizando la clase de disponibilidad, el conductor es capaz de registrar cuándo y con que carro hará su trabajo
Prueba Válida: (”Lunes”,”AAA111”)-> todas son llaves válidas y no tiene primary key porque es SA
Prueba Inválida(”AAA111”)->El dia es NN entonces no cuenta como valido

RF6: La función está pensada de tal manera que a la hora de cambiar una disponibilidad, el conductor reemplaza automáticamente la de ese día y hora, de esta manera no genera errores.
Prueba Válida: (”Lunes”,”AAA111”)-> todas son llaves válidas y no tiene primary key porque es SA
Prueba Inválida(”AAA111”)->El dia es NN entonces no cuenta como valido

RF7: El usuario es capaz de crear puntos e implementarlos en sus servicios además de ponerles sus nombres.
Prueba Válida: (”111”,”220”,”cr15b#192-65”,”llegada”, “111111”,”102914”)-> todas son llaves válidas y no tiene primary key porque es SA
Prueba Inválida(”111”,,”cr15b#192-65”,”llegada”, “111111”,”102914”)->La latitud es NN entonces no cuenta como válido

RF8: Los métodos permiten al usuario de servicio generar un servicio con los respectivos parámetros, además de almacenarlos.
Prueba Válida: (”111”,”220”,”cr15b#192-65”,”llegada”, “111111”,”102914”)-> todas son llaves válidas y no tiene primary key porque es SA
Prueba Inválida:(”111”,,”cr15b#192-65”,”llegada”, “111111”,”102914”)->La latitud es NN entonces no cuenta como válido

RF9: Cuando se termina el viaje se cambia el parámetro de disponibilidad provisional, el cual no tiene que guardarse en base de datos, y el servicio realizado se guarda.
Prueba Válida:("S001", "C123", "COND456", "ABC123", "BOG01", "PAS", "TAR001", 35, 12.5, "2025-08-31 14:30:00")-> todas son llaves válidas y los campos Duración, Distancia y FechaHoraInicio cumplen con la restricción NN, por lo tanto se registra correctamente.
Prueba Inválida:(“11111”,"S001", "C123", "COND456", "ABC123", "BOG01", "PAS", "TAR001", 35, 12.5, "2025-08-31 14:30:00")->Intenta agregar PK por lo que esta ma

RF10/11: Cada usuario es capaz de dejar una revisión que quedará enlazada a otro usuario y así mismo obligatoriamente, utilizando el método de la clase padre.
Prueba Válida: (4,”epico”,”cliente”,”102914”)-> todas son llaves válidas y no tiene primary key porque es SA
Prueba Inválida: (,”epico”,””,”102914”)->Sin puntuacion o tipo revisor, este no es valido.

REQUERIMIENTOS FUNCIONALES DE CONSULTA

Mostrar estadísticas de uso de los recursos de **ALPESCAB**, entre las que se encuentran:

RFC1 - CONSULTAR EL HISTÓRICO DE TODOS LOS SERVICIOS PEDIDOS POR UN USUARIO
Dado un usuario de servicios específico, se debe mostrar una lista de todos los servicios junto con toda su información relevante.

RFC2 - MOSTRAR LOS 20 USUARIOS CONDUCTORES QUE MÁS SERVICIOS HAN PRESTADO EN LA APLICACIÓN
Esta consulta permite identificar qué conductores han prestado la mayor cantidad de servicios en la aplicación, indicando además de los datos del conductor, el número de servicios que han prestado.

RFC3 - MOSTRAR EL TOTAL DE DINERO OBTENIDO POR USUARIOS CONDUCTORES PARA CADA UNO DE SUS VEHÍCULOS, DISCRIMINADO POR SERVICIOS

Esta consulta permite visualizar a un conductor el dinero ganado por todos los servicios prestados. Acá debe aparecer un agregado por los servicios diferentes y el total de dinero obtenido (tras deducir las comisiones que cobra **ALPESCAB**)

RFC4 - MOSTRAR LA UTILIZACIÓN DE SERVICIOS DE ALPESCAB EN UNA CIUDAD DURANTE UN RANGO DE FECHAS INDICADO.
Esta consulta permite visualizar las estadísticas de uso de todos los servicios y niveles de estos para un rango de fechas inicial y final, junto con el porcentaje del total al que corresponde el número de servicios. Esto se muestra siempre comenzando por el servicio más usado al menos utilizado.

RFC1: Mediante de llaves foráneas en servicios, se pueden relacionar y consultar la información necesaria de sus pedidos

RFC2: Teniendo la relación con servicio y cliente, se debería contar las veces que aparece la llave foránea de un usuarios, si estas se ordenan se tendrían los 20 mayores.

RFC3: En la tabla servicio se tiene el costo, por lo cual por la misma relación que tiene con el cliente, esta se puede consultar de manera eficiente utilizando el método de la clase.

RFC4: Utilizando la tabla tipo de servicio se facilita la obtención de la utilidad dependiendo del tipo de servicio.

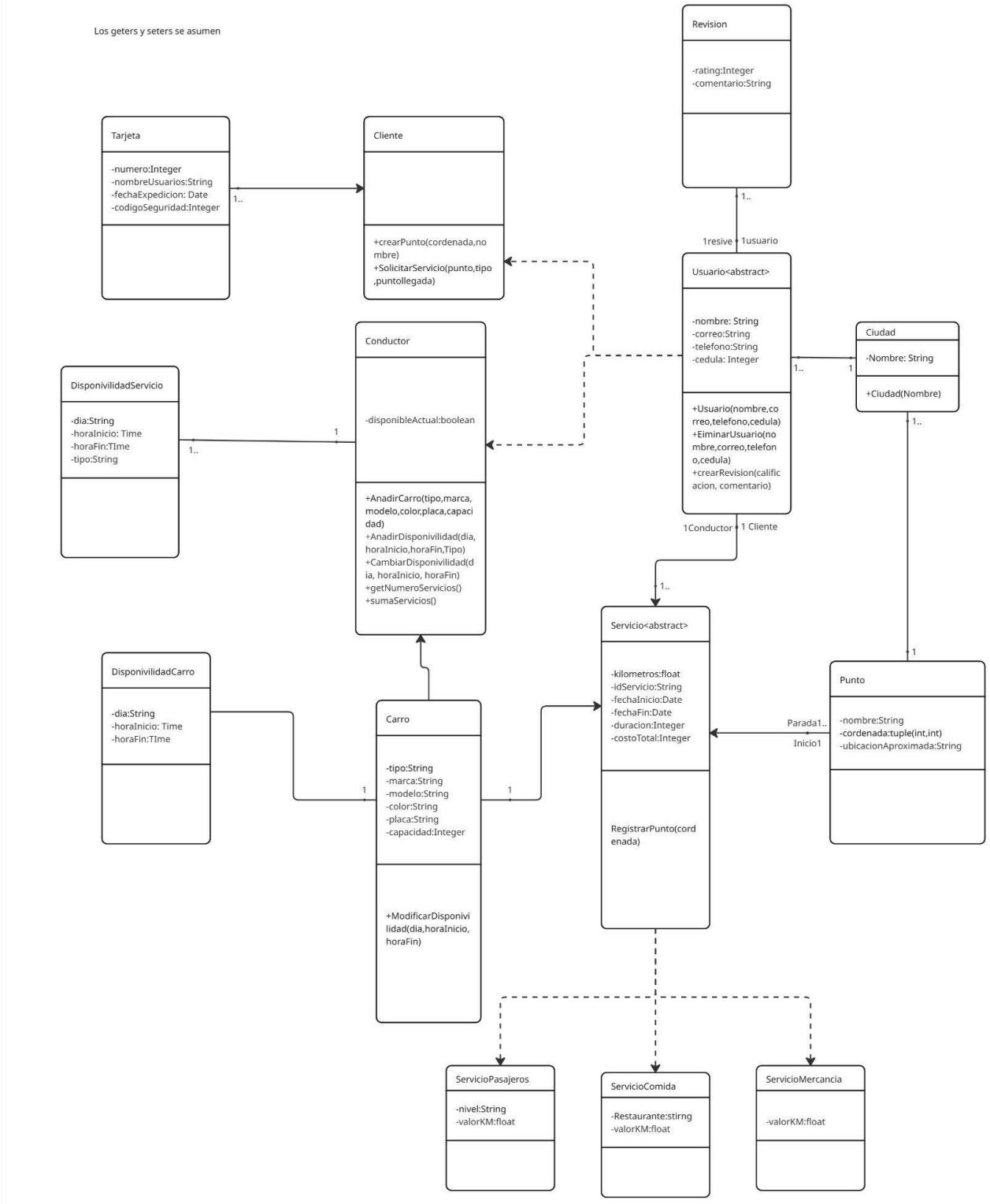
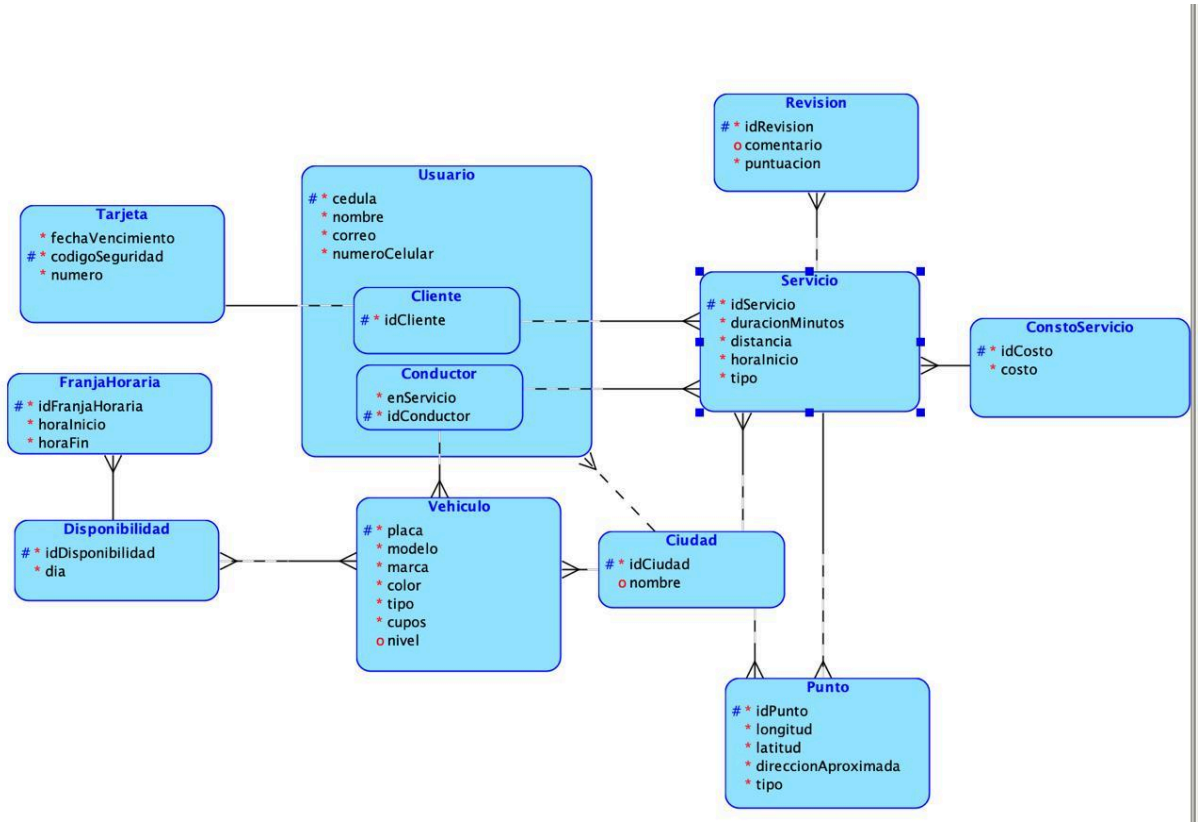

Los geters y seters se asumen
Revision
-rating:Integer
-comentario:String
Tarjeta
-numero:Integer
-nombreUsuarios:String
-fechaExpedicion: Date
-codigoSeguridad:Integer
Cliente
+crearPunto(cordenada,nombre)
+SolicitarServicio(punto,tipo,puntollegada)
1..
1resive
1usuario
Usuario<abstract>
-nombre: String
-correo:String
-telefono:String
-cedula: Integer
+Usuario(nombre,correo,telefono,cedula)
+EiminarUsuario(nombre,correo,telefono,cedula)
+crearRevision(calificacion, comentario)
Ciudad
-Nombre: String
+Ciudad(Nombre)
Conductor
-disponibleActual:boolean
+AnadirCarro(tipo,marca,modelo,color,placa,capacidad)
+AnadirDisponivilidad(dia,horaInicio,horaFin,Tipo)
+CambiarDisponivilidad(dia, horaInicio, horaFin)
+getNumeroServicios()
+sumaServicios()
DisponivilidadServicio
-dia:String
-horaInicio: Time
-horaFin:TIme
-tipo:String
1
1..
1Conductor
1 Cliente
Servicio<abstract>
-kilometros:float
-idServicio:String
-fechaInicio:Date
-fechaFin:Date
-duracion:Integer
-costoTotal:Integer
RegistrarPunto(cordenada)
Parada1..
Inicio1
Punto
-nombre:String
-cordenada:tuple(int,int)
-ubicacionAproximada:String
DisponivilidadCarro
-dia:String
-horaInicio: Time
-horaFin:TIme
Carro
-tipo:String
-marca:String
-modelo:String
-color:String
-placa:String
-capacidad:Integer
+ModificarDisponivilidad(dia,horaInicio,horaFin)
ServicioPasajeros
-nivel:String
-valorKM:float
ServicioComida
-Restaurante:stirng
-valorKM:float
ServicioMercancia
-valorKM:float

Diagrama bark



Verificación de formas normales - Todas las entidades cumplen hasta BCNF

Entidad Usuario:

Usuario

Cédula	Nombre	Correo	Celular	IdCiudad
PK	NN	NN	$CK_{(3000000000..3999999999)}$	$FK_{Ciudad.idCiudad}$

Nombre de la relación
Atributos/Dominio
Restricciones
Tuplas

1NF: Se cumple debido a que los atributos son atómicos y no hay multivalores.
2NF: Todos los atributos dependen únicamente de la cédula. Por lo tanto se cumple.
3NF: Las dependencias son:
Cedula -> (Nombre,Correo,Celular,IdCiudad)
No hay ninguna dependencia entre atributos no clave. Por lo tanto se cumple.
BCNF: Como cédula es una superclave, se cumple.

Entidad Cliente:

Cliente

IdCliente	Cédula	NumeroTarjeta
PK	$FK_{Usuario.Cedula}$	$FK_{Tarjeta.NumeroTarjeta}$

Nombre de la relación
Atributos/Dominio
Restricciones
Tuplas

- 1NF: Se cumple ya que no debería haber más de una cédula por cliente y más de una tarjeta asignada a un solo cliente.

- 2NF: Se cumple ya que tanto el número de la tarjeta como la cédula depende del idCliente.
- 3NF: Las dependencias son:
  IdCliente -> (Cédula, NumeroTarjeta)
  Por lo tanto cumple con 3NF porque IdCliente es primary Key.
- BCNF: Al ser IdCliente llave primaria, se cumple.

Entidad Conductor:

Conductor

IdConductor	Cédula	EnServicio
PK	**$FK_{Usuario.Cedula}$**	**$CK_{[True/False]}$**

Nombre de la relación
Atributos/Dominio
Restricciones
Tuplas

- 1NF: Se cumple porque un conductor solo puede tener una cédula y o esta en servicio o no esta en servicio.
- 2NF: Ambos atributos dependen unicamente del conductor, por lo tanto se cumple.
- 3NF: Las dependencias son:
  IdConductor -> (Cedula, EnServicio)
  Por lo tanto se cumple la 3NF.
- BCNF: Se cumple ya que IdConductor es una PrimaryKey.

Entidad Vehiculo

Vehiculo

Placa	Marca	Modelo	Tipo	Color	Cupos	Nivel	IdCiudad	IdConductor
PK	**NN**	**NN**	**NN**	**NN**	**$CK_{[1..10]}$**		**$FK_{Ciudad.idCiudad}$**	**$FK_{Conductor.IdConductor}$**

- 1NF: Se cumple ya que todos los atributos son atómicos.
- 2NF: Se cumple ya que no hay dependencias parciales y la placa es la llave primaria.
- 3NF: Como en el contexto cada atributo depende de la placa (siendo la placa el identificador único de cada vehículo), entonces se cumple.
- BCNF: Como la placa es la llave primaria y de esta depende todos, pues si cumple el BCNF.

Entidad Ciudad:

Ciudad

IdCiudad	Nombre
PK, SA	**NN**

Nombre de la relación
Atributos/Dominio
Restricciones
Tuplas

- 1NF: Se cumple ya que un idCiudad sólo puede tener un nombre.
- 2NF: Se cumple ya que no hay ningún tipo de dependencia parcial.
- 3NF: Se cumple ya que la dependencia es:
  IdCiudad -> Nombre
  por lo tanto no existen dependencias transitivas y se cumple la 3NF.
- BCNF: El único determinante es idCiudad, y es PK por lo tanto se cumple.

Entidad Servicio:

Servicio

IdServicio	IdCliente	IdConductor	Placa	idCiudad	IdTipoServicio	IdCosto	Duración	Distancia	FechaHoralnicio
PK, SA	$FK_{Cliente.IdCliente}$	$FK_{Conductor.IdConductor}$	$FK_{Vehiculo.Placa}$	$FK_{Ciudad.idCiudad}$	$FK_{TipoServicio.IdTipoSe}$	$FK_{CostoServicio.IdCost}$	NN	NN	NN

- 1NF: Se cumple ya que un servicio contiene uno de cada uno de estos atributos.
- 2NF: Se cumple ya que no hay ningún tipo de dependencia parcial entre los atributos.
- 3NF: Los atributos no claves no dependen de otro atributo no clave.
- BCNF: Todos los atributos dependen de IdServicio, por lo tanto se cumple.

Entidad TipoServicio:

TipoServicio

IdTipoServicio	Nivel	Descripcion	Tarifa
PK, SA		NN	NN

- 1NF: Se cumple ya que un tipo de servicio no debería tener, ni más de un nivel, ni de una descripción, ni de una tarifa.
- 2NF: Se cumple porque no hay dependencias parciales entre los atributos que no son IdTipoServicio.
- 3NF: Las dependencias son:
  idTipoServicio -> (Nivel,Descripcion,Tarifa)
  por lo tanto se cumple 3NF.
- BCNF: IdTipoServicio es una PK, por lo tanto cumple.

Entidad Punto:

Punto

Id punto	Longitud	Latitud	DireccionAproximada	Tipo	IdCiudad	IdServicio
PK, SA	NN	NN	NN	NN	$FK_{Ciudad.idCiudad}$	$FK_{Servicio.IdServicio}$

- 1NF**:** Cumple, ya que todos los atributos son atómicos y no hay repetición de grupos de datos.
- 2NF: La PK es idPunto y nada tiene dependencias parciales, por lo tanto cumple.
- 3NF: Ningún atributo no clave depende de otro no clave, por lo tanto cumple.
- BCNF: Todas las dependencias funcionales tienen como determinante una clave candidata, por lo tanto cumple.

Entidad Revisión:

Revisión

Id revisión	Puntuación	Comentario	TipoRevisor	IdServicio
PK,SA	NN, $CK_{[0..5]}$		NN	$FK_{Servicio.IdServicio}$

- 1NF**:** Cumple ya que la revisión contiene un único valor por cada atributo.
- 2NF: Cumple por que no hay dependencias parciales.
- 3NF: Cumple porque:
  IdRevision -> (Puntuación, Comentario, TipoRevisor, IdServicio)
- BCNF: Los determinantes son claves primarias por lo tanto cumple.

Entidad Disponibilidad:

Disponibilidad

IdDisponibilidad	Dia	Placa
PK, SA	NN	$FK_{Vehiculo.Placa}$

- 1NF: Cumple porque todos los atributos son atómicos (día es un valor único, placa también).
- 2NF: La clave primaria es IdDisponibilidad, así que no hay dependencias parciales, por lo tanto cumple
- 3NF: Cumple porque:
- IdDisponibilidad -> (Dia, Placa) y por lo tanto se cumple.
- BCNF: IdDisponibilidad es la PK por lo tanto cumple.

Entidad FranjaHoraria:

Franja horaria

IdFranjaHoraria	IdDisponibilidad	HoraInicio	HoraFin
PK, SA	$FK_{Disponibilidad.IdDisponibilid}$	NN, $CK_{[00:00..23:59]}$	NN, $CK_{[00:00..23:59]}$

- 1NF: Todos los atributos deberían ser valores únicos, osea cumple.
- 2NF: No hay dependencias parciales, por lo tanto se cumple.
- 3NF: Cumple porque:
- IdFranjaHoraria -> (IdDisponibilidad , HoraInicio, HoraFin).
- BCNF: IdFranjaHoraria es una llave primaria, osea cumple.

Entidad Tarjeta:

Tarjeta

NumeroTarjeta	Nombre	Fecha vencimiento	Codigo seguridad
PK	**NN**	**NN**	**NN**

- 1NF: Una tarjeta tiene un único nombre, una única fecha de vencimiento y un único código, por lo tanto cumple.
- 2NF: Todo depende unicamente del numero de la tarjeta, por ende cumple.
- 3NF: Cumple porque:
- NumeroTarjeta-> (Nombre,FechaVencimiento,CodigoSeguridad).
- BCNF: La PK determina todo del resto de atributos.

Entidad CostoServicio:

CostoServicio

IdCosto	Costo
PK	**NN**

- 1NF: No pueden haber varios costos por un solo IdCosto. Se cumple.
- 2NF: Las dependencias son:
  IdCosto -> Costo
  Por ende cumple tanto 2NF como 3NF.
- 3NF: Cumple
- BCNF: El idCosto identifica únicamente a ese costo. Se cumple.