

Informe Proyecto 1

Integrantes: Andrés Bolívar, Valeria Martínez, Cristian Rugeles

Nombre del usuario en Oracle:

ISIS2304B15202520

Contraseña:

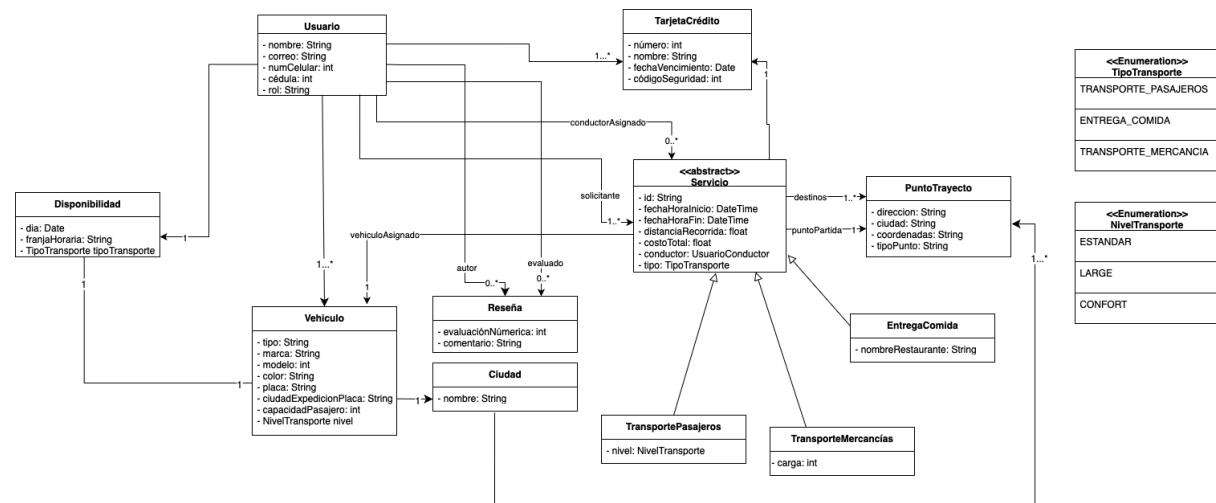
ztgzigVGZMlw

1. Análisis y Revisión del Modelo de Datos

Para esta entrega, se ha llevado a cabo una revisión exhaustiva del modelo de datos conceptual y relacional presentado en el diseño inicial. Tomando como punto de partida el feedback recibido, se realizaron ajustes significativos con el objetivo de aumentar la claridad, reducir la complejidad, eliminar ambigüedades y asegurar que todas las reglas de negocio del caso AlpesCab queden representadas de manera precisa y robusta.

A continuación, se describen los modelos resultantes y las modificaciones clave que se implementaron.

El diagrama de clases UML es el plano conceptual de nuestra aplicación. El modelo inicial fue refactorizado para corregir omisiones y simplificar su estructura, resultando en un diseño más limpio y flexible.



Modificaciones Realizadas y Justificación:

- Unificación de la Entidad Usuario:** El cambio más significativo fue la eliminación de las clases especializadas **UsuarioConductor** y **UsuarioDeServicio**. Se reemplazaron por una única clase **Usuario** que contiene un atributo **rol('CONDUCTOR' o 'SERVICIOS')**. Esta decisión simplifica drásticamente el modelo, reduce la redundancia y aumenta la flexibilidad, permitiendo que en el futuro un usuario pueda desempeñar ambos roles sin necesidad de alterar la estructura.

- **Claridad en la Relación Servicio - PuntoTrayecto:** Atendiendo al feedback, se eliminó la relación ambigua de uno a muchos entre Servicio y PuntoTrayecto. En su lugar, se crearon dos asociaciones explícitas y nombradas:
 - Una relación **puntoPartida** (un servicio tiene un único punto de partida).
 - Una relación **destinos** (un servicio tiene uno o varios puntos de destino). Esto elimina cualquier duda sobre el rol de cada punto en un trayecto.
- **Contextualización de la Reseña:** La clase Reseña fue rediseñada para tener el contexto correcto. Ahora se asocia directamente con el Servicio al que pertenece. Adicionalmente, se establecieron dos relaciones hacia Usuario, con los roles **autor** y **evaluado**, dejando perfectamente claro quién califica a quién después de un viaje específico.
- **Vínculo entre Servicio y Vehiculo:** Se añadió la asociación faltante entre la clase Servicio y Vehiculo. Esto es fundamental para el sistema, ya que permite registrar qué vehículo específico fue utilizado para prestar cada servicio, un requisito indispensable para el historial y las consultas de ganancias.
- **Registro del Pago del Servicio:** Se incorporó una relación directa entre Servicio y TarjetaCredito para registrar con qué medio de pago específico se abonó cada transacción. Esto fortalece la integridad transaccional del modelo, permitiendo auditorías y un historial de pagos preciso para el usuario.

A continuación, se verifica el cumplimiento de las formas normales para cada tabla del modelo relacional propuesto para AlpesCab. El objetivo es garantizar que el diseño minimice la redundancia y prevenga anomalías de datos, asegurando que cada relación se encuentre en la **Forma Normal de Boyce-Codd (FNBC)**.

(Documento .xlsx adjunto para conservar calidad)

Primera Forma Normal (1FN)

- **Requisito:** La 1FN exige que todos los atributos de una tabla sean atómicos, es decir, que cada celda contenga un único valor y que no existan grupos de repetición.
- **Verificación:** Todas las tablas en nuestro modelo (USUARIO, TARJETA_CREDITO, VEHICULO, SERVICIO, DESTINOS_SERVICIO, RESEÑA, etc.) cumplen con la Primera Forma Normal. Cada columna ha sido diseñada para almacenar un valor atómico (un número, una cadena de texto, una fecha), y no existen atributos multivaluados ni grupos repetitivos.
- **Conclusión:** El modelo completo se encuentra en **1FN**.

Segunda Forma Normal (2FN)

- **Requisito:** La 2FN exige que la tabla esté en 1FN y que todos sus atributos no clave dependan funcionalmente de la clave primaria **completa**. Esta forma es relevante principalmente para tablas con claves primarias compuestas.
- **Verificación:**

- **Tablas con Clave Primaria Simple:** La mayoría de nuestras tablas (USUARIO, VEHÍCULO, SERVICIO, RESEÑA, PUNTO_TRAYECTO, CIUDAD, etc.) tienen una clave primaria simple (un solo atributo). Por definición, no pueden tener dependencias parciales, por lo que cumplen automáticamente con 2FN.
- **Tabla con Clave Primaria Compuesta (DESTINOS_SERVICIO):**
 - **Clave Primaria:** {id_servicio, id_punto_destino}.
 - **Atributos no clave:** {orden}.
 - El atributo orden depende de la clave completa, ya que indica la secuencia de una parada (id_punto_destino) dentro de un viaje específico (id_servicio). No depende solo de una parte de la clave. Por lo tanto, cumple con 2FN.
- **Conclusión:** El modelo completo se encuentra en **2FN**.

Tercera Forma Normal (3FN)

- **Requisito:** La 3FN exige que la tabla esté en 2FN y que no existan dependencias transitivas, es decir, que ningún atributo no clave dependa funcionalmente de otro atributo no clave.
- **Verificación:**
 - **USUARIO:** Los atributos nombre, correo, num_celular y rol dependen directamente de la cedula y no entre sí. Cumple 3FN.
 - **VEHICULO:** Atributos como marca, modelo y color son características intrínsecas del vehículo identificado por la placa. No dependen de otros atributos no clave. Cumple 3FN.
 - **SERVICIO:** Todos los atributos, incluidas las claves foráneas como cedula_solicitante, placa_vehiculo o numero_tarjeta_pago, dependen directamente del id del servicio. No hay dependencias entre ellos (por ejemplo, la placa del vehículo no determina la tarjeta con la que se pagó). Cumple 3FN.
 - **RESEÑA:** Los atributos evaluacion_numerica y comentario dependen directamente del id de la reseña, y no entre sí. Cumple 3FN.
 - **Resto de Tablas:** Las demás tablas del modelo (CIUDAD, PUNTO_TRAYECTO, etc.) han sido diseñadas para que sus atributos no clave dependan exclusivamente de su clave primaria, evitando así cualquier dependencia transitiva.
- **Conclusión:** El modelo completo se encuentra en **3FN**.

Forma Normal de Boyce-Codd (FNBC)

- **Requisito:** La FNBC es una versión más estricta de la 3FN. Exige que para cada dependencia funcional no trivial $X \rightarrow Y$, X sea una superllave de la tabla.

- **Verificación:** Para violar la FNBC, una tabla necesitaría tener múltiples claves candidatas que se superponen. Se revisa cada tabla de nuestro modelo:
 - En todas las tablas del modelo (USUARIO, VEHICULO, SERVICIO, etc.), la **única clave candidata identificada es la propia clave primaria** que hemos definido (cedula, placa, id, etc.).
 - No existen otras claves candidatas compuestas o alternativas.
 - Por lo tanto, para cualquier dependencia funcional en nuestras tablas, el determinante (el lado izquierdo de la dependencia) es siempre la clave primaria. Por definición, la clave primaria es una superllave.
- **Conclusión:** El modelo completo se encuentra en **FNBC**.

Conclusión Final de Normalización

El modelo de datos relacional propuesto cumple con la **Forma Normal de Boyce-Codd (FNBC)**. Cada tabla ha sido diseñada para minimizar la redundancia y prevenir anomalías de actualización, inserción y borrado, asegurando que todos los atributos dependan "de la clave, la clave completa y nada más que la clave". Este nivel de normalización garantiza una base de datos robusta y escalable, capaz de implementar todos los requerimientos funcionales del caso de estudio de AlpesCab.

2. Desarrollo de la aplicación en Java Spring

El objetivo fue integrar consultas SQL en un backend desarrollado con Java Spring para exponerlas como servicios REST, de manera que puedan ser consumidas fácilmente desde Postman o cualquier frontend.

Se implementaron los reportes RFC1, RFC2, RFC3 y RFC4, siguiendo una arquitectura limpia y organizada.

Arquitectura utilizada

La solución se estructuró en cuatro capas principales:

- **DTOs:** Son las clases que representan los datos que se devolverán al cliente. Cada consulta tiene su propio DTO que contiene solo la información necesaria para ese reporte.
- **Repositorio:** Contiene la lógica SQL. Aquí se ejecutan las consultas directamente sobre la base de datos Oracle utilizando JdbcTemplate.
- **Servicio:** Actúa como intermediario entre el controlador y el repositorio. Llama a los métodos del repositorio y retorna los resultados.
- **Controlador:** Expone los endpoints REST que permiten acceder a cada reporte a través de URLs tipo GET.

Clases Principales de Java Spring

El código fuente, ubicado en el repositorio, está organizado en paquetes que reflejan la arquitectura descrita. Las clases más importantes son:

- **com.sistrans.controller:**
 - **UsuarioController.java:** Expone los endpoints para registrar usuarios (/servicios y /conductores).
 - **ServicioController.java:** Maneja la lógica para solicitar (/solicitar) y finalizar (/{{id}}/finalizar) servicios.
 - **ReporteController.java:** Expone los endpoints para los Requerimientos Funcionales de Consulta (RFCs), como /reportes/top-conductores (RFC2).
- **com.sistrans.service:**
 - **UsuarioService.java:** Contiene la lógica transaccional para crear un usuario de servicios junto con su tarjeta de crédito.
 - **ServicioService.java:** Orquesta la compleja lógica de negocio para la asignación de un nuevo servicio.
 - **ReporteService.java:** Prepara los datos para los reportes solicitados por el ReporteController.
- **com.sistrans.repository:**
 - **UsuarioRepository.java:** Ejecuta las sentencias INSERT para las tablas USUARIO y TARJETA_CREDITO.
 - **ServicioRepository.java:** Contiene el SQL para buscar conductores disponibles e insertar nuevos servicios.
 - **RfcRepository.java:** Almacena las consultas SQL complejas para los RFCs, utilizando RowMapper para transformar los resultados en DTOs.
- **com.sistrans.dto y com.sistrans.entity:**
 - Se utiliza el patrón **DTO (Data Transfer Object)** para separar la representación de los datos en la API de la estructura interna de la base de datos (entity), mejorando la seguridad y la flexibilidad del sistema.

Se configuró la conexión a Oracle en el archivo application.properties con la URL, usuario, contraseña y el driver correspondiente.

Se añadieron las dependencias necesarias en pom.xml:

- spring-boot-starter-web para exponer endpoints.
- spring-boot-starter-jdbc para ejecutar SQL.
- spring-boot-starter-data-jpa para manejar consultas complejas.
- ojdbc8 para conectar con Oracle Database.

3. Escenarios de pruebas con Postman:

Resultados de las pruebas RFC1:

```
GET http://localhost:8080/api/reportes/rfc1/2009?limite=100 Send

Params • Authorization Headers (6) Body Scripts • Settings Cookies

Body Cookies Headers (5) Test Results (1/1) ⚡ 200 OK • 835 ms • 1.62 KB • 🔍 Save Response ⚡

{} JSON ▾ Preview Visualize ▾

1 [ 
2 {
3     "id_servicio": 45,
4     "fecha_inicio": "2025-08-18T01:43:18",
5     "fecha_fin": null,
6     "costo_total": 20000.0,
7     "tipo_servicio": "TRANSPORTE_PASAJEROS",
8     "punto_partida": "Dirección Controlada 6 - Bogotá",
9     "punto_destino": "Dirección Controlada 11 - Bogotá",
10    "nombre_conductor": "Conductor 45",
11    "placa_vehiculo": "AUT045"
12 },
13 {
14     "id_servicio": 44,
15     "fecha_inicio": "2025-08-18T00:43:18",
16     "fecha_fin": null,
17     "costo_total": 19000.0,
18     "tipo_servicio": "TRANSPORTE_PASAJEROS",
19     "punto_partida": "Dirección Controlada 5 - Bogotá",
20     "punto_destino": "Dirección Controlada 10 - Bogotá",
21     "nombre_conductor": "Conductor 44",
22     "placa_vehiculo": "AII044"
23 ]
```

Resultados de las pruebas RFC2:

HTTP RFC / Top Conductores por Número de Servicios

GET http://localhost:8080/api/reportes/rfc2/top-conductores?limite=20

Params • Authorization Headers (6) Body Scripts Settings Cookies

Body (radio buttons): none (selected), form-data, x-www-form-urlencoded, raw, binary, GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results 200 OK 237 ms 2.29 KB Save Response

{ } JSON ▾ ▶ Preview Visualize

```
[{"cedula": "1014", "nombre": "Conductor 14", "correo": "conductor14@alpescab.com", "total_servicios_prestados": 10}, {"cedula": "1018", "nombre": "Conductor 18", "correo": "conductor18@alpescab.com", "total_servicios_prestados": 10}, {"cedula": "1020", "nombre": "Conductor 20", "correo": "conductor20@alpescab.com", "total_servicios_prestados": 10}]
```

Resultados de las pruebas RFC3:

HTTP RFC / Ganancias por conductor

GET <http://localhost:8080/api/reportes/rfc3/ganancias-conductor/1001> Send

Params Authorization Headers (6) Body Scripts Settings Cookies

This request does not have a body

Body Cookies Headers (5) Test Results 200 OK 124 ms 257 B Save Response

{ } JSON ▾ Preview Visualize

```
1 [  
2   {  
3     "placa_vehiculo": "AUT001",  
4     "tipo_servicio": "TRANSPORTE_PASAJEROS",  
5     "ganancia_total": 96000.0  
6   }  
7 ]
```

Resultados de las pruebas RFC4:

HTTP RFC / Utilizacion de servicios en rango de fechas

GET <http://localhost:8080/api/reportes/rfc4/uso-servicios?desde=2025-01-01T00:00:00&hasta=2025-12...> Send

Params • Authorization Headers (6) Body Scripts Settings Cookies

This request does not have a body

Body Cookies Headers (5) Test Results 200 OK 129 ms 259 B Save Response

{ } JSON ▾ Preview Visualize

```
1 [  
2   {  
3     "tipo_servicio": "TRANSPORTE_PASAJEROS",  
4     "numero_de_viajes": 1000,  
5     "porcentaje_del_total": 100.0  
6   }  
7 ]
```

PRUEBAS DE LOS RF1 a RF11

RF1: REGISTRAR UNA CIUDAD

Prueba: Registrar Ciudad Nueva (exitoso)

- Descripción: Se realiza una petición **POST** para registrar una nueva ciudad ("Cartagena") que no existe en la base de datos.
- Resultado Esperado: La API responde con **Status: 201 Created**.
- Evidencia (Postman):

HTTP RF - Proyecto Sistrans / RF1 - Registrar Ciudad / Registrar Ciudad Nueva (exitoso)

POST {{base}}/ciudades

Params Authorization Headers (9) Body Scripts Settings

Query Params

	Key	Value	Descrip
	Key	Value	Descrip

Body Cookies Headers (5) Test Results 201 Created 64 Raw Preview Visualize 1 Ciudad creada exitosamente

Prueba: Registrar Ciudad - Barranquilla

- Descripción: Se realiza una petición **POST** para registrar una segunda ciudad ("Barranquilla").
- Resultado Esperado: La API responde con **Status: 201 Created**.

- **Evidencia (Postman):**

The screenshot shows the Postman interface for a 'Create City' request. The URL is `HTTP RF - Proyecto Sistrans / RF1 - Registrar Ciudad / Registrar Ciudad - Barranquilla`. The method is POST, and the endpoint is `{{base}}/ciudades`. The 'Params' tab is selected, showing a single parameter 'Key' with 'Value'. In the response section, the status is '201 Created' and the body contains the message '1 Ciudad creada exitosamente'.

Prueba: Fallo - Ciudad Duplicada

- **Descripción:** Se intenta registrar una ciudad ("Bogotá") que ya fue insertada previamente.
- **Resultado Esperado:** La API debe rechazar la petición y responder con un error del cliente, **Status: 400 Bad Request**, indicando que el recurso ya existe.
- **Evidencia (Postman):**

The screenshot shows the Postman interface for a 'Create City' request. The URL is `HTTP RF - Proyecto Sistrans / RF1 - Registrar Ciudad / Fallo - Ciudad duplicada (Bogotá ya existe)`. The method is POST, and the endpoint is `{{base}}/ciudades`. The 'Params' tab is selected, showing a single parameter 'Key' with 'Value'. In the response section, the status is '400 Bad Request' and the body contains the error message '1 PreparedStatementCallback; SQL [INSERT INTO CIUDAD (nombre) VALUES (?)]; ORA-00001: unique constraint (ISIS2304B15202520. SYS_C001492805) violated'.

RF2: REGISTRAR UN USUARIO DE SERVICIOS

Prueba: Crear Usuario de Servicios (exitoso)

- **Descripción:** Se envía una petición **POST** a **/usuarios/servicios**. Esta operación transaccional crea un usuario con rol 'SERVICIOS' y su tarjeta de crédito asociada.
- **Resultado Esperado:** La API debe responder con **Status: 201 Created**, confirmando la creación de ambos registros.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF2 - Registrar Usuario de Servicios / Registrar Usuario Servicios - Nuevo (exitoso)

POST {{base}}/usuarios/servicios

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 201 Created 75 ms

Raw Preview Visualize

1 Usuario de servicios creado exitosamente

HTTP RF - Proyecto Sistrans / RF2 - Registrar Usuario de Servicios / Registrar Usuario Servicios - María

POST {{base}}/usuarios/servicios

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 201 Created

Raw Preview Visualize

1 Usuario de servicios creado exitosamente

Prueba: Fallo - Datos Incompletos (sin tarjeta de crédito)

- **Descripción:** Se intenta registrar un nuevo usuario de servicios sin dar información de la tarjeta de crédito.
- **Resultado Esperado:** La API debe rechazar la petición y responder con un error **Status: 400 Bad Request**.
- **Evidencia (Postman):**

The screenshot shows a Postman request for 'RF - Proyecto Sistrans / RF2 - Registrar Usuario de Servicios / Fallo - Datos incompletos (sin tarjeta de crédito)'. The method is POST to {{base}}/usuarios/servicios. The Body tab is selected, showing a JSON payload with fields: cedula (50003), nombre (Usuario Incompleto), correo (incompleto@test.com), and celular (3001111111). The response status is 400 Bad Request, with the message: 'La información de la tarjeta de crédito es requerida'.

Prueba: Fallo - Cédula Duplicada

- **Descripción:** Se intenta registrar un nuevo usuario de servicios utilizando una cédula que ya existe en la base de datos.
- **Resultado Esperado:** La API debe rechazar la petición y responder con un error **Status: 400 Bad Request**.
- **Evidencia (Postman):**

The screenshot shows a Postman request for 'RF - Proyecto Sistrans / RF2 - Registrar Usuario de Servicios / Fallo - Cédula duplicada'. The method is POST to {{base}}/usuarios/servicios. The Body tab is selected, showing a JSON payload with a single field: Key. The response status is 400 Bad Request, with the message: 'PreparedStatementCallback; SQL [INSERT INTO USUARIO (cedula, nombre, correo, celular, rol) VALUES (?, ?, ?, ?, ?)]; ORA-00001: unique constraint (ISIS2304B1520250.SYS_C001492810) violated'.

RF3: REGISTRAR UN USUARIO CONDUCTOR

Prueba: Crear Usuarios Conductores (exitoso)

- **Descripción:** Se envía una petición **POST** a **/usuarios/conductores** para dar de alta a un nuevo usuario con el rol de 'CONDUCTOR'.
- **Resultado Esperado:** La API debe responder con **Status: 201 Created**.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF3 - Registrar Usuario Conductor / Registrar Conductor - Nuevo (exitoso)

POST | {{base}}/usuarios/conductores

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results | ⏱ 201 Created 72 ms

Raw ▾ Preview Visualize |

1 Usuario conductor creado exitosamente

HTTP RF - Proyecto Sistrans / RF3 - Registrar Usuario Conductor / Registrar Conductor - Ana

POST | {{base}}/usuarios/conductores

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results | ⏱ 201 Created 32 ms

Raw ▾ Preview Visualize |

1 Usuario conductor creado exitosamente

Prueba: Fallo - Cédula Duplicada

- **Descripción:** Se intenta registrar un nuevo conductor utilizando una cédula que ya existe en la base de datos.
- **Resultado Esperado:** La API debe responder con un error **Status: 400 Bad Request.**
- **Evidencia (Postman):**

The screenshot shows a Postman interface for a POST request to `http://{{base}}/usuarios/conductores`. The 'Params' tab is selected. A single query parameter 'Key' is defined with the value 'Value'. The 'Send' button is visible at the top right. Below the request area, the response section shows a red '400 Bad Request' status. The raw response body contains the following SQL error message:
1 PreparedStatementCallback; SQL [INSERT INTO USUARIO (cedula, nombre, correo, celular, rol) VALUES (?, ?, ?, ?, ?)]; ORA-00001:
unique constraint (ISIS2304B15202520.SYS_C001492810) violated
2

Prueba: Fallo - Correo Duplicado

- **Descripción:** Se intenta registrar un nuevo conductor utilizando un correo que ya existe en la base de datos.
- **Resultado Esperado:** La API debe responder con un error **Status: 400 Bad Request.**
- **Evidencia (Postman):**

The screenshot shows a Postman interface for a POST request to `http://{{base}}/usuarios/conductores`. The 'Params' tab is selected. A single query parameter 'Key' is defined with the value 'Value'. The 'Send' button is visible at the top right. Below the request area, the response section shows a red '400 Bad Request' status. The raw response body contains the following SQL error message:
1 PreparedStatementCallback; SQL [INSERT INTO USUARIO (cedula, nombre, correo, celular, rol) VALUES (?, ?, ?, ?, ?)]; ORA-00001:
unique constraint (ISIS2304B15202520.SYS_C001492811) violated
2

The screenshot shows a Postman interface for a POST request to `http://{{base}}/usuarios/conductores`. The 'Params' tab is selected. A single query parameter 'Key' is defined with the value 'Value'. The 'Send' button is visible at the top right. Below the request area, the response section shows a red '400 Bad Request' status. The raw response body contains the following SQL error message:
1 PreparedStatementCallback; SQL [INSERT INTO USUARIO (cedula, nombre, correo, celular, rol) VALUES (?, ?, ?, ?, ?)]; ORA-00001:
unique constraint (ISIS2304B15202520.SYS_C001492811) violated
2

RF4: REGISTRAR UN VEHÍCULO

Prueba: Crear Vehículo (exitoso)

- **Descripción:** Se realiza una petición **POST** al endpoint **/vehiculos** para registrar distintos tipos de nuevos vehículos, asociándolos a la cédula de un conductor previamente creado.
- **Resultado Esperado:** La API debe responder con **Status: 201 Created**.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF4 - Registrar Vehículo / Registrar Vehículo Estándar (exitoso)

POST | {{base}}/vehiculos

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 {
2   "placa": "XYZ001",
3   "tipo": "carro",
4   "marca": "Chevrolet",
5   "modelo": 2022,
6   "color": "Blanco",
7   "ciudad_placa": "Bogotá",
8   "capacidad_pasajero": 4,
9   "nivel_transporte": "ESTANDAR",
10  "cedula_dueno": 30001
11 }
```

Body Cookies Headers (5) Test Results | ⚡

201 Created • 31 ms

Raw Preview Visualize

1 Vehículo creado exitosamente

HTTP RF - Proyecto Sistrans / RF4 - Registrar Vehículo / Registrar Vehículo Confort

POST | {{base}}/vehiculos

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results | ⚡

201 Created • 32 ms

Raw Preview Visualize

1 Vehículo creado exitosamente

HTTP RF - Proyecto Sistrans / RF4 - Registrar Vehículo / **Registrar Vehículo Large**

POST {{base}}/vehiculos

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

201 Created

Raw Preview Visualize

1 Vehiculo creado exitosamente

HTTP RF - Proyecto Sistrans / RF4 - Registrar Vehículo / **Registrar Vehículo - Motocicleta**

POST {{base}}/vehiculos

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 {
2   "placa": "XYZ004",
3   "tipo": "motocicleta",
4   "marca": "Yamaha",
5   "modelo": 2023,
6   "color": "Azul",
7   "ciudad_placa": "Bogotá",
8   "capacidad_pasajero": 1,
9   "nivel_transporte": "ESTANDAR",
10  "cedula_dueno": 30002
11 }
```

Body Cookies Headers (5) Test Results

201 Created • 27 ms

Raw Preview Visualize

1 Vehiculo creado exitosamente

Prueba: Fallo - Placa Duplicada

- **Descripción:** Se intenta registrar un vehículo con una placa que ya existe en la base de datos.
- **Resultado Esperado:** La API debe responder con un error **Status: 400 Bad Request.**
- **Evidencia (Postman):**

The screenshot shows a Postman request to 'POST {{base}}/vehiculos'. The 'Body' tab contains a JSON payload with a single key 'placa' set to 'XYZ999'. The response status is '400 Bad Request' with a message indicating a unique constraint violation (ORA-00001) because the plate 'XYZ999' already exists. The 'Headers' tab shows '(4)' results.

Prueba: Fallo - Conductor no existe

- **Descripción:** Se intenta registrar un vehículo asignándolo a una **cedula_dueño** que no corresponde a ningún usuario.
- **Resultado Esperado:** La API debe responder con un error **Status: 400 Bad Request.**
- **Evidencia (Postman):**

The screenshot shows a Postman request to 'POST {{base}}/vehiculos'. The 'Body' tab contains a JSON payload with a single key 'placa' set to 'XYZ999'. The response status is '400 Bad Request' with a message indicating an integrity constraint violation (ORA-02291) because the conductor with cedula '999999' does not exist. The 'Headers' tab shows '(4)' results.

RF5 y RF6: GESTIONAR DISPONIBILIDAD

Prueba: Crear Disponibilidades (exito)

- **Descripción:** Se registra una franja horaria para un vehículo a través de una petición **POST** a **/disponibilidad**.
- **Resultado Esperado:** La API responde **Status: 201 Created**.
- **Evidencia (Postman):**

RF - Proyecto Sistrans / RF5 - Registrar Disponibilidad / **Crear Disponibilidad - Transporte Pasajeros (exito)**

POST {{base}} /disponibilidad

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 201 Created 67 ms

Raw Preview Visualize

1 Disponibilidad creada con ID: 25

RF - Proyecto Sistrans / RF5 - Registrar Disponibilidad / **Crear Disponibilidad - Tarde**

POST {{base}} /disponibilidad

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results 201 Created 64 ms

Raw Preview Visualize

1 Disponibilidad creada con ID: 26

HTTP RF - Proyecto Sistrans / RF5 - Registrar Disponibilidad / **Crear Disponibilidad - Entrega Comida**

POST {{base}} /disponibilidad

Params Authorization Headers (9) Body Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results

201 Created • 87 ms

Preview

1 Disponibilidad creada con ID: 27

HTTP RF - Proyecto Sistrans / RF5 - Registrar Disponibilidad / **Crear Disponibilidad - Transporte Mercancías**

POST {{base}} /disponibilidad

Params Authorization Headers (9) Body Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results

201 Created • 61 ms

Preview

1 Disponibilidad creada con ID: 28

Prueba: Crear Disponibilidades sin conductor (Fallo)

- **Descripción:** Se registra una franja horaria para un vehículo a través de una petición **POST** a **/disponibilidad** sin un conductor asignado.
- **Resultado Esperado:** La API responde **Status: 400 Bad Request**.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF5 - Registrar Disponibilidad / Fallo - Sin conductor

POST [`{base}/disponibilidad`](#)

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (4) Test Results

400 Bad Request ≈ 30 ms

Raw Preview Debug with AI

1 La cédula del conductor es requerida para crear disponibilidad

Prueba: Modificar Disponibilidades (exitoso)

- **Descripción:** Se actualiza una franja horaria existente a través de una petición **PUT** a **/disponibilidad/{id}**.
- **Resultado Esperado:** La API responde **Status: 200 OK**.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF6 - Modificar Disponibilidad / Modificar Disponibilidad (exitoso)

PUT [`{base}/disponibilidad/1`](#)

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

200 OK ≈ 55 ms

Raw Preview Visualize

1 Disponibilidad modificada exitosamente

HTTP RF - Proyecto Sistrans / RF6 - Modificar Disponibilidad / **Modificar Disponibilidad - Cambiar día**

PUT ▼ {{base}} /disponibilidad/1

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results | ⏱ 200 OK • 52 ms

Raw ▾ Preview Visualize | ▾

```
1 Disponibilidad modificada exitosamente
```

Prueba: Fallo - Superposición de Horarios

- **Descripción:** Se intenta registrar una segunda disponibilidad para el mismo vehículo en una franja horaria que se cruza con una ya existente.
- **Resultado Esperado:** La lógica de negocio debe detectar el conflicto y devolver un error **Status: 409 Conflict**.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF6 - Modificar Disponibilidad / **Fallo - Modificación con superposición**

PUT ▼ {{base}} /disponibilidad/1

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (4) Test Results | ⏱ 400 Bad Request • 42 ms

Raw ▾ Preview ⚡ Debug with AI | ▾

```
1 Ya existe una disponibilidad con horarios superpuestos en el día indicado
```

RF7: REGISTRAR PUNTO GEOGRÁFICO

Prueba: Crear Puntos Geográficos (exitoso)

- **Descripción:** Se crean nuevos puntos geográficos, como una dirección frecuente, a través de una petición **POST** a **/puntos**.
- **Resultado Esperado:** La API responde **Status: 201 Created**.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF7 - Registrar Punto Geográfico / **Crear Punto - Recogida (exitoso)**

POST {{base}}/puntos

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results 201 Created 61 ms

Raw ▾ Preview Visualize ▾

1 Punto de trayecto creado con ID: 21

HTTP RF - Proyecto Sistrans / RF7 - Registrar Punto Geográfico / **Crear Punto - Destino**

POST {{base}}/puntos

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results 201 Created 57 ms

Raw ▾ Preview Visualize ▾

1 Punto de trayecto creado con ID: 22

HTTP RF - Proyecto Sistrans / RF7 - Registrar Punto Geográfico / Crear Punto - Con nombre establecimiento

POST {{base}} /puntos

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results | ⏱ 201 Created • 49 ms •

Raw ▾ Preview Visualize ▾

1 Punto de trayecto creado con ID: 23

HTTP RF - Proyecto Sistrans / RF7 - Registrar Punto Geográfico / Crear Punto - Restaurante

POST {{base}} /puntos

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results | ⏱ 201 Created • 52 ms

Raw ▾ Preview Visualize ▾

1 Punto de trayecto creado con ID: 24

RF8: SOLICITAR SERVICIO

Prueba: Solicitar Servicios (exitoso)

- **Descripción:** Se simula la solicitud de un viaje a través de un **POST** al endpoint **/servicios/solicitar**. La operación debe encontrar un conductor disponible y crear los registros correspondientes.
- **Resultado Esperado:** La API responde con **Status: 201 Created** y devuelve un JSON con la información del servicio creado.
- **Evidencia (Postman):**

HTTP RF - Proyecto Sistrans / RF8 - Solicitar Servicio / **Solicitar Servicio Estándar (exitoso)**

POST **((base)) /servicios/solicitar**

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

201 Created 145 ms

{ } JSON Visualize

```
1 {  
2   "id": 1677,  
3   "fecha_hora_inicio": null,  
4   "fecha_hora_fin": null,  
5   "distancia": null,  
6   "costo_total": 20000.0,  
7   "tipo": "TRANSPORTE_PASAJEROS",  
8   "cedula_solicitante": 50001,  
9   "cedula_conductor": 1019,  
10  "placa_vehiculo": "AUT019",  
11  "id_punto_partida": 1,  
12  "tarjeta_credito": "4111111111111111"  
13 }
```

HTTP RF - Proyecto Sistrans / RF8 - Solicitar Servicio / **Solicitar Servicio Confort**

POST **((base)) /servicios/solicitar**

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

201 Created 106 ms

{ } JSON Visualize

```
1 {  
2   "id": 1678,  
3   "fecha_hora_inicio": null,  
4   "fecha_hora_fin": null,  
5   "distancia": null,  
6   "costo_total": 20000.0,  
7   "tipo": "TRANSPORTE_PASAJEROS",  
8   "cedula_solicitante": 50002,  
9   "cedula_conductor": 1083,  
10  "placa_vehiculo": "AUT083",  
11  "id_punto_partida": 3,  
12  "tarjeta_credito": "4222222222222222"  
13 }
```

HTTP RF - Proyecto Sistrans / RF8 - Solicitar Servicio / **Solicitar Servicio - Entrega Comida**

POST [`{{base}}/servicios/solicitar`](#)

Params Authorization Headers (9) Body Scripts Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results |

201 Created • 119 ms

{ } JSON ▾ ▶ Preview Visualize ▾

```
1 {  
2   "id": 1679,  
3   "fecha_hora_inicio": null,  
4   "fecha_hora_fin": null,  
5   "distancia": null,  
6   "costo_total": 20000.0,  
7   "tipo": "ENTREGA_COMIDA",  
8   "cedula_solicitante": 50001,  
9   "cedula_conductor": 1061,  
10  "placa_vehiculo": "AUT061",  
11  "id_punto_partida": 7,  
12  "tarjeta_credito": "4111111111111111"  
13 }
```

HTTP RF - Proyecto Sistrans / RF8 - Solicitar Servicio / **Solicitar con Usuario Poblado**

POST [`{{base}}/servicios/solicitar`](#)

Params Authorization Headers (9) **Body** Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** ▾

	Key	Value
1	"cedula_solicitante"	2001
2	"id_punto_partida"	5
3	"ids_puntos_destino"	[10]
4	"tipo_servicio"	"TRANSPORTE_PASAJEROS"
5	"numero_tarjeta"	"4444444444442001"
6	"nivel_transporte"	"ESTANDAR"
7		
8		

Body Cookies Headers (5) Test Results |

201 Created • 110 ms

{ } JSON ▾ ▶ Preview Visualize ▾

```
1 {  
2   "id": 1680,  
3   "fecha_hora_inicio": null,  
4   "fecha_hora_fin": null,  
5   "distancia": null,  
6   "costo_total": 20000.0,  
7   "tipo": "TRANSPORTE_PASAJEROS",  
8   "cedula_solicitante": 2001,  
9   "cedula_conductor": 1050,  
10  "placa_vehiculo": "AUT050",  
11  "id_punto_partida": 5,  
12  "tarjeta_credito": "4444444444442001"  
13 }
```

Prueba: Solicitar Servicios con usuario inexistente (fallo)

- **Descripción:** Se simula la solicitud de un viaje a través de un **POST** al endpoint **/servicios/solicitar**. La operación no es exitosa pues el usuario 99999 no existe
- **Resultado Esperado:** La API responde con **Status: 400 Bad Request**
- **Evidencia (Postman):**

The screenshot shows a Postman request configuration. The method is POST, the URL is `http://{{base}}/servicios/solicitar`, and the body contains a single key 'Key' with the value 'Value'. The response status is 400 Bad Request.

```
1 PreparedStatementCallback; SQL [INSERT INTO SERVICIO (fecha_hora_inicio, fecha_hora_fin, distancia, costo_total, tipo, cedula_solicitante, cedula_conductor, placa_vehiculo, id_punto_partida, tarjeta_credito) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)]; ORA-02291: integrity constraint (ISIS2304B15202520.FK_SERVICIO_SOLICITANTE) violated - parent key not found
```

RF9: FINALIZAR SERVICIO

Prueba: Finalizar un Servicio (exitoso)

- **Descripción:** Se actualiza el estado de un servicio en curso a "Finalizado" a través de una petición **PUT** al endpoint **/servicios/{id}/finalizar**.
- **Resultado Esperado:** La API responde con **Status: 200 OK** y un mensaje de confirmación.
- **Evidencia (Postman):**

The screenshot shows a Postman request configuration. The method is PUT, the URL is `http://{{base}}/servicios/201/finalizar`, and the body contains a single key 'Key' with the value 'Value'. The response status is 200 OK.

```
1 Servicio finalizado exitosamente
```

RF - Proyecto Sistrans / RF9 - Registrar Viaje (Finalizar Servicio) / **Finalizar Servicio - Viaje Largo**

PUT

▼

`{{base}}/servicios/202/finalizar`

Params

Authorization

Headers (9)

Body

Scripts

Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results |

200 OK

• 39 ms •

Raw ▾

▷ Preview

Visualize ▾

1 Servicio finalizado exitosamente

RF - Proyecto Sistrans / RF9 - Registrar Viaje (Finalizar Servicio) / **Finalizar Servicio - Viaje Corto**

PUT

▼

`{{base}}/servicios/203/finalizar`

Params

Authorization

Headers (9)

Body

Scripts

Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results |

200 OK

• 38 ms • 19

Raw ▾

▷ Preview

Visualize ▾

1 Servicio finalizado exitosamente

Prueba: Finalizar un Servicio que no existe (fallo)

- **Descripción:** Se actualiza el estado de un servicio inexistente en curso a "Finalizado" a través de una petición **PUT** al endpoint **/servicios/{id}/finalizar**.
- **Resultado Esperado:** La API responde con **Status: 400 Bad Request**
- **Evidencia (Postman):**

The screenshot shows a Postman request configuration for a PUT method to the endpoint {{base}}/servicios/99999/finalizar. The 'Body' tab is selected. In the raw body section, there is a single line of JSON: { "status": "Finalizado" }. The response tab shows a 400 Bad Request status with the message "El servicio con ID 99999 no existe".

RF10 y RF11: DEJAR RESEÑA

Prueba: Crear Reseña (Pasajero a Conductor)

- **Descripción:** Se envía una petición **POST** a **/resenas** donde un pasajero califica a un conductor por un servicio ya completado. (Hay más escenarios de este RF)
- **Resultado Esperado:** La API responde con **Status: 201 Created**.
- **Evidencia (Postman):**

The screenshot shows a Postman request configuration for a POST method to the endpoint {{base}}/resenas. The 'Body' tab is selected, containing a JSON object with 'calificacion' and 'comentario' fields. The response tab shows a 201 Created status with the message "Reseña creada exitosamente".

Prueba: Crear Reseña (Conductor a Pasajero)

- **Descripción:** Se envía una petición **POST** a **/resenas** donde un conductor califica a un pasajero por un servicio completado. (Hay mas escenarios para este RF)
- **Resultado Esperado:** La API responde con **Status: 201 Created.**
- **Evidencia (Postman):**

The screenshot shows a Postman interface for a POST request to `POST {{base}}/resenas`. The 'Body' tab is selected. In the 'Params' section, there is a single entry with 'Key' and 'Value'. The response status is 201 Created, and the message is 'Reseña creada exitosamente'.

Prueba: Fallo - Calificación Invalida

- **Descripción:** Se intenta registrar una reseña con una puntuación invalida para un **id_servicio**.
- **Resultado Esperado:** La API responde con **Status: 400 Bad Request**
- **Evidencia (Postman):**

The screenshot shows a Postman interface for a POST request to `POST {{base}}/resenas`. The 'Body' tab is selected. In the 'Params' section, there is a single entry with 'Key' and 'Value'. The response status is 400 Bad Request, and the message is 'La evaluación debe ser un número entre 0 y 5'.

Prueba: Fallo - Servicio no existe

- Descripción: Se intenta registrar una reseña para un `id_servicio` que no existe en la base de datos.
- Resultado Esperado: La API responde con `Status: 400 Bad Request`
- Evidencia (Postman):

The screenshot shows a Postman interface for a 'POST' request to `RF - Proyecto Sistrans / RF11 - Revisión Conductor a Usuario / Fallo - Servicio no existe`. The 'Body' tab is selected, showing a single key 'Key' with the value 'Value'. The 'Test Results' tab shows a red '400 Bad Request' status with a response message: '1 El usuario no fue el conductor de este servicio'. The 'Headers' tab shows 9 headers. The 'Cookies' and 'Raw' tabs are also visible.

5. Balance del Plan de Pruebas

El plan de pruebas ejecutado permitió validar de manera integral la funcionalidad de la aplicación AlpesCab. Se logró una cobertura completa de los requerimientos especificados, aunque también se identificaron áreas para futuras mejoras.

Logros

1. Cobertura Completa de Requerimientos: Se implementaron y probaron con éxito todos los 11 Requerimientos Funcionales (RF1-RF11) y los 4 Requerimientos Funcionales de Consulta (RFC1-RFC4). Cada endpoint de la API fue validado utilizando Postman.
2. Validación de Reglas de Negocio: Las pruebas de falla demostraron que la aplicación maneja correctamente las reglas de negocio críticas. Esto incluye la prevención de registros con datos duplicados (cédulas, placas), la validación de superposición de horarios para la disponibilidad de conductores y el manejo de relaciones de integridad.
3. Integridad Transaccional: Se verificó el correcto funcionamiento de las operaciones transaccionales. En pruebas como el registro de un usuario de servicios (RF2), se

confirmó que si la inserción de la tarjeta de crédito falla, la **creación del usuario se revierte (ROLLBACK)**, manteniendo la base de datos en un estado consistente.

4. **Población de Datos:** Se generó y utilizó con éxito un script para poblar la base de datos con un volumen de datos significativo (100 conductores, 200 pasajeros, 1000 servicios), lo cual fue fundamental para realizar pruebas realistas sobre las consultas de los RFCs.

Pendientes y Oportunidades de Mejora

1. Lógica de Asignación de Conductores (RF8): La implementación actual para la búsqueda de conductores disponibles es funcional pero simplificada. El sistema asigna uno de los primeros conductores que cumple con los requisitos básicos, pero no incluye algoritmos avanzados de optimización (como buscar el más cercano geográficamente) o balanceo de carga.
2. Validación de Disponibilidad (RF5 y RF6): Aunque se implementó la validación para prevenir la superposición de horarios, esta se basa en una lógica que podría ser más detallada. Una mejora futura sería manejar rangos de tiempo con mayor precisión (a nivel de minutos) y considerar posibles excepciones o pausas dentro de una franja de disponibilidad.
3. Cálculo de Tarifas y Distancias: Para esta entrega, el cálculo del costo total y la distancia se manejó con valores simulados o fijos. Una implementación completa requeriría la integración con una API de mapas (como Google Maps API) para calcular distancias y tarifas dinámicas en tiempo real.

En resumen, el plan de pruebas fue exitoso en validar que la aplicación cumple con todos los requisitos funcionales solicitados para esta entrega, sentando una base de software robusta y bien estructurada sobre la cual se pueden construir funcionalidades más complejas en el futuro.

Diseño de sentencias SQL

RFC1: Consultar el Histórico de Servicios de un Usuario

- **Objetivo:** Mostrar un listado completo de todos los servicios que ha solicitado un usuario específico, incluyendo detalles del conductor, vehículo y trayecto.
- **Tablas Usadas:**
 - SERVICIO (s): Tabla principal que contiene la información de cada viaje.
 - USUARIO (u_conductor): Se une para obtener el nombre del conductor.
 - VEHÍCULO (v): Se une para obtener los detalles del vehículo usado.
 - PUNTO_TRAYECTO (pt_partida): Se une para obtener la dirección del punto de partida.
- **Atributos de JOIN:**
 - s.cedula_conductor = u_conductor.cedula
 - s.placa_vehiculo = v.placa
 - s.id_punto_partida = pt_partida.id_punto

- **Tipo de JOIN y Justificación:**

- Se utiliza LEFT JOIN para unir USUARIO y VEHICULO. Esto es crucial porque un servicio puede haber sido solicitado pero aún no tener un conductor o vehículo asignado. Usar INNER JOIN ocultaría estos servicios del historial.
- Se utiliza INNER JOIN para PUNTO_TRAYECTO, ya que es obligatorio que todo servicio tenga un punto de partida registrado.

```

SELECT
    s.id AS id_servicio,
    s.fecha_hora_inicio,
    s.costo_total,
    s.tipo AS tipo_servicio,
    pt_partida.direccion AS punto_de_partida,
    u_conductor.nombre AS nombre_conductor,
    v.placa
FROM
    SERVICIO s
LEFT JOIN
    USUARIO u_conductor ON s.cedula_conductor = u_conductor.cedula
LEFT JOIN
    VEHICULO v ON s.placa_vehiculo = v.placa
INNER JOIN
    PUNTO_TRAYECTO pt_partida ON s.id_punto_partida = pt_partida.id_punto
WHERE
    s.cedula_solicitante = 2001
ORDER BY
    s.fecha_hora_inicio DESC;

```

Resultado de la Consulta | Resultado de la Consulta 1 | SQL | Todas las Filas Recuperadas: 5 en 0,047 segundos

ID_SERVICIO	FECHA_HORA_INICIO	COSTO_TOTAL	TIPO_SERVICIO	PUNTO_DE_PARTIDA	NOMBRE_CONDUCTOR	PLACA
1	5 16/08/25 09:43:18,000000000 AM	20000	TRANSPORTE_PASAJEROS	Dirección Controlada 6	Conductor 5	AUT005
2	4 16/08/25 08:43:18,000000000 AM	19000	TRANSPORTE_PASAJEROS	Dirección Controlada 5	Conductor 4	AUT004
3	3 16/08/25 07:43:18,000000000 AM	18000	TRANSPORTE_PASAJEROS	Dirección Controlada 4	Conductor 3	AUT003
4	2 16/08/25 06:43:18,000000000 AM	17000	TRANSPORTE_PASAJEROS	Dirección Controlada 3	Conductor 2	AUT002
5	1 16/08/25 05:43:18,000000000 AM	16000	TRANSPORTE_PASAJEROS	Dirección Controlada 2	Conductor 1	AUT001

RFC2: Mostrar los 20 Conductores con Más Servicios Prestados

- **Objetivo:** Generar un ranking con los 20 conductores que han completado más servicios, mostrando sus datos y el total de servicios.
- **Tablas Usadas:**
 - SERVICIO (s): Se usa para contar el número de servicios asociados a cada conductor.
 - USUARIO (u): Se une para obtener el nombre y el correo del conductor.
- **Atributos de JOIN:**
 - s.cedula_conductor = u.cedula
- **Tipo de JOIN y Justificación:**
 - Se utiliza INNER JOIN porque solo nos interesan los usuarios que efectivamente han realizado servicios. Los conductores que no hayan completado ningún viaje no son relevantes para este ranking.

```

SELECT
    u.cedula,
    u.nombre,
    u.correo,
    COUNT(s.id) AS total_servicios_prestados
FROM
    USUARIO u
INNER JOIN
    SERVICIO s ON u.cedula = s.cedula_conductor
WHERE
    u.rol = 'CONDUCTOR'
GROUP BY
    u.cedula, u.nombre, u.correo
ORDER BY
    total_servicios_prestados DESC
FETCH FIRST 20 ROWS ONLY;

```

Resultado de la Consulta x Resultado de la Consulta 1 x

SQL | Todas las Filas Recuperadas: 20 en 0,034 segundos

CEDULA	NOMBRE	CORREO	TOTAL_SERVICIOS_PRESTADOS
1020	Conductor 20	conductor20@alpescab.com	10
4	1043 Conductor 43	conductor43@alpescab.com	10
5	1051 Conductor 51	conductor51@alpescab.com	10
6	1056 Conductor 56	conductor56@alpescab.com	10
7	1062 Conductor 62	conductor62@alpescab.com	10
8	1069 Conductor 69	conductor69@alpescab.com	10
9	1072 Conductor 72	conductor72@alpescab.com	10
10	1073 Conductor 73	conductor73@alpescab.com	10
11	1079 Conductor 79	conductor79@alpescab.com	10
12	1084 Conductor 84	conductor84@alpescab.com	10
13	1087 Conductor 87	conductor87@alpescab.com	10
14	1093 Conductor 93	conductor93@alpescab.com	10
15	1096 Conductor 96	conductor96@alpescab.com	10
16	1004 Conductor 4	conductor4@alpescab.com	10
17	1006 Conductor 6	conductor6@alpescab.com	10
18	1011 Conductor 11	conductor11@alpescab.com	10
19	1017 Conductor 17	conductor17@alpescab.com	10
20	1018 Conductor 18	conductor18@alpescab.com	10

RFC3: Mostrar Ganancias por Vehículo y Tipo de Servicio

- **Objetivo:** Calcular el total de dinero ganado por un conductor, desglosado por cada uno de sus vehículos y por el tipo de servicio prestado.
- **Tablas Usadas:**
 - SERVICIO (s): Contiene toda la información necesaria: el costo, el vehículo (placa_vehiculo) y el tipo de servicio.
- **Atributos de JOIN:** No se requieren JOINs.
- **Tipo de JOIN y Justificación:** No aplica. Toda la información reside en la tabla SERVICIO. Se utiliza una cláusula GROUP BY para agregar los resultados.

```

SELECT
    s.placa_vehiculo,
    s.tipo AS tipo_servicio,
    SUM(s.costo_total * 0.60) AS ganancia_total
FROM
    SERVICIO s
WHERE
    s.cedula_conductor = 1001
GROUP BY
    s.placa_vehiculo,
    s.tipo
ORDER BY
    s.placa_vehiculo,
    ganancia_total DESC;

```

Resultado de la Consulta | SQL | Todas las Filas Recuperadas: 1 en 0,017 segundos

PLACA_VEHICULO	TIPO_SERVICIO	GANANCIA_TOTAL
1 AUT001	TRANSPORTE_PASAJEROS	96000

RFC4: Utilización de Servicios en una Ciudad y Rango de Fechas

- **Objetivo:** Mostrar estadísticas de uso de los diferentes tipos de servicio en una ciudad y un periodo de tiempo determinados, ordenados por popularidad y mostrando el porcentaje que representa cada uno.
- **Tablas Usadas:**
 - SERVICIO (s): Contiene el tipo de servicio y la fecha de inicio.
 - PUNTO_TRAYECTO (pt): Se une para filtrar los servicios que partieron desde la ciudad de interés.
- **Atributos de JOIN:**
 - s.id_punto_partida = pt.id_punto
- **Tipo de JOIN y Justificación:**
 - Se utiliza INNER JOIN porque solo nos interesan los servicios que tienen un punto de partida válido y registrado en la ciudad que se está consultando.

```

SELECT
    s.tipo AS tipo_servicio,
    COUNT(*) AS numero_de_viajes,
    ROUND((COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2) AS porcentaje_del_total
FROM
    SERVICIO s
INNER JOIN
    PUNTO_TRAYECTO pt ON s.id_punto_partida = pt.id_punto
WHERE
    pt.ciudad = 'Bogotá'
    AND s.fecha_hora_inicio BETWEEN TO_TIMESTAMP('2025-09-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS') AND TO_TIMESTAMP('2025-09-30 23:59:59', 'YYYY-MM-DD HH24:MI:SS')
GROUP BY
    s.tipo
ORDER BY
    numero_de_viajes DESC;

```

Resultado de la Consulta | SQL | Todas las Filas Recuperadas: 1 en 0,026 segundos

TIPO_SERVICIO	NUMERO_DE_VIAJES	PORCENTAJE_DEL_TOTAL
1 TRANSPORTE_PASAJEROS	621	100