Informe Proyecto 1 Implementación

Este informe documenta la implementación del esquema de base de datos, la población de datos y el diseño de las consultas RFC1–RFC4 para el caso de estudio ALPESCAB (según el enunciado). Se explica cómo se transformó el modelo relacional a sentencias CREATE TABLE, se describen las decisiones de diseño (PK, FK, constraints), se detalla la estrategia de población con bloques PL/SQL y se liga cada consulta RFC con las tablas y atributos usados (explicando el tipo de JOIN elegido).

La actualización del informe original se realizó y se encuentra en el archivo que tiene el mismo nombre, pero con actualizado y contiene correcciones de lo que faltaba en el documento original, lo principal era la justificación de pasar de UML al Modelo Relacional y también la modificación de las tablas en el Modelo Relacional tal que todas las tablas son BCNF.

1. De modelo relacional a tablas SQL

El paso de un modelo relacional (fichas / E/R / UML) a sentencias SQL se realizó siguiendo las reglas estándar:

1. **Cada relación del modelo relacional → una tabla SQL.**
    - Ejemplo: la relación Usuario del modelo se mapeó a la tabla Usuario(cedula, nombre, correo, ...).
2. **Claves primarias (PK):**
    - Si la relación tenía un identificador natural fiable (ej. placa para Vehiculo) se usó como PK natural.
    - Para relaciones sin identificador natural o donde se requiere un surrogado por simplicidad, se creó una columna autogenerada (GENERATED BY DEFAULT AS IDENTITY) como PK (ej. idPunto, idDisponibilidad).
3. **Claves foráneas (FK):**
    - Las asociaciones 1:N se implementaron colocando la FK en el lado N. Ej.: Vehiculo(conductor_cedula) referencia Usuario(cedula).
    - Las asociaciones M:N se normalizaron en tablas intermedias; ejemplo: ServicioPunto(servicio_id, punto_id) para modelar varios puntos por servicio.
4. **Subclases / herencia:**
    - Se usó **tabla por subclase con PK compartida** cuando la subclase tenía atributos específicos (p. ej. UsuarioConductor, UsuarioCliente) para evitar columnas NULL y mantener integridad con la tabla padre Usuario.
5. **Restricciones y check:**
    - Se añadieron CHECK para validar reglas de dominio (ej. CHECK (horaFin > horaInicio) en Disponibilidad, CHECK(rating BETWEEN 0 AND 5) en Rating).
6. **Atributos derivados:**
    - Atributos como valorServicio (distancia * tarifa) se pueden marcar como columna calculada o mantenerse como columna normal actualizada por trigger. En este esquema se usó una columna virtual en Servicio (valorServicio GENERATED ALWAYS AS (distancia * tarifa) VIRTUAL) con la observación de que, si la tarifa depende del subtipo o del nivel, es preferible calcularlo en la aplicación o en triggers que integren subtipo y servicio.

2. Mapeo tabla por tabla (resumen)

A continuación, un resumen conciso para las tablas clave (por qué se creó, PK y FK relevantes).

- **Ciudad** — idCiudad (PK autogenerado), nombre (UNIQUE). Necesaria para ubicar puntos y vehículos.
- **PuntoGeografico** — idPunto (PK), direccion, latitud, longitud, ciudad (FK→Ciudad). Modela puntos de partida/llegada y establecimientos.
- **Usuario** — cedula (PK), datos personales. Padre de subclases UsuarioCliente y UsuarioCoductor.
- **UsuarioCliente / UsuarioCoductor** — PK cedula y FK a Usuario para modelar roles y datos específicos.
- **Vehiculo** — placa (PK), cedula (FK→Usuario), ciudadExpedicion (FK→Ciudad). Identifica vehículos asociados a conductores.
- **Disponibilidad** — idDisponibilidad (PK), placa (FK→Vehiculo), fecha, horaInicio, horaFin. Controla franjas horarias disponibles.
- **Servicio** — idServicio (PK), tipo, distancia, tarifa, valorServicio (virtual), horaInicio, horaFin, clienteCedula (FK→Usuario), conductorCedula (FK→Usuario), placa (FK→Vehiculo).
- **ServicioPunto** — idServicioPunto (PK), idServicio (FK→Servicio), idPunto (FK→PuntoGeografico). Resuelve N:M entre servicios y puntos.
- **Subtipos de servicio** (TransportePasajeros, TransporteMercancias, EntregaDomicilio) — tablas con PK idServicio y FK a Servicio(idServicio) para atributos específicos de cada subtipo.
- **Rating (Revision)** — idComentario (PK), idServicio (FK), revisor y revisado (FK→Usuario), rating con CHECK.
- **TarjetaCredito** — numero (PK), cedula (FK→Usuario). Se recomienda almacenar solo la máscara/token y evitar CVV.

3. Población de tablas — uso de loops y control de volúmenes

3.1 Estrategia general

Para poblar datos de prueba se creó un script PL/SQL con bloques BEGIN ... FOR i IN 1..N LOOP ... END LOOP; END; / que inserta filas en cada tabla respetando el orden de dependencias: primero Ciudad y PuntoGeografico, luego Usuario, Vehiculo, Servicio, subtables y finalmente ServicioPunto, Rating, TarjetaCredito.

Ventajas del enfoque con loops:

- Permite generar muchos registros de forma reproducible y rápida sin escribir miles de INSERT a mano.
- Al variar el límite del FOR i IN 1..N controlamos exactamente la cardinalidad final (por ejemplo pasar de 100 a 1000 conductores cambiando 100 → 1000).
- Se puede usar DBMS_RANDOM para diversificar valores (fechas, tarifas, coordenadas).

Manejo de dependencias y duplicados: cada bloque maneja excepciones EXCEPTION WHEN DUP_VAL_ON_INDEX THEN NULL; para que el script sea *idempotente* (puedes re-ejecutarlo varias veces sin fallar por duplicados). Además, los scripts insertan en orden que asegura que las FKs referencien filas existentes.

3.2 Cómo cumplir los requisitos mínimos (100 conductores, 200 pasajeros, 5 viajes por pasajero)

- **100 conductores diferentes:** Al poblar Usuario se asigna tipo = 'CONDUCTOR' a al menos 100 filas (ej. FOR i IN 1..300 y setear tipo en función de i para garantizar >=100 conductores). Luego, en UsuarioCoductor insertamos las primeras 100 cédulas con ese tipo.

- **200 pasajeros inscritos:** Mismas técnicas — generar usuarios con tipo = 'CLIENTE' y poblar UsuarioCliente para 200 usuarios.
- **Media de 5 viajes por pasajero:** si tenemos P pasajeros inscritos (por ejemplo 200), generamos V = P * 5 servicios con clienteCedula asignada cíclicamente a los pasajeros. En un loop se puede hacer cliente := 'Cedula' || MOD(i-1, P) + 1 para que cada pasajero reciba viajes rotativamente hasta llegar a 5 en promedio.

  Ejemplo sencillo (pseudocódigo PL/SQL):

```
-- Asumiendo 200 pasajeros en UsuarioCliente (Cedula1..Cedula200)
FOR i IN 1..(200*5) LOOP
  cliente := 'Cedula' || MOD(i-1,200)+1; -- asigna cliente cíclicamente
  INSERT INTO Servicio (...) VALUES (..., cliente, ...);
END LOOP;
```

  Con esto aseguramos la **media de 5 viajes por pasajero**.

4. Documentación de las consultas RFC1–RFC4

A continuación, se documenta, para cada consulta RFC, las tablas utilizadas, los atributos empleados en los joins, la sentencia SQL y una breve justificación del tipo de JOIN elegido.

RFC1 — Histórico de servicios de un usuario

Objetivo: dado un cliente, mostrar la lista de todos sus servicios y atributos relacionados (incluyendo tarifas según subtipo).

Tablas usadas y atributos (joins):

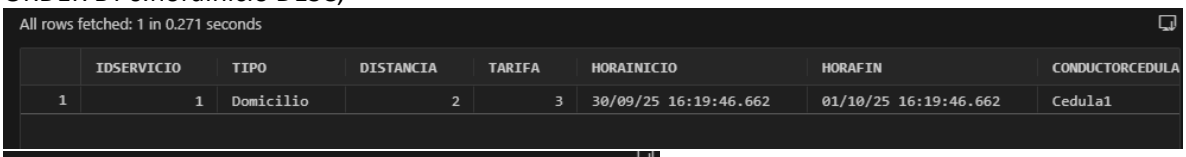
- Servicio s — s.idServicio, s.tipo, s.distancia, s.tarifa, s.horaInicio, s.horaFin, s.placa, s.conductorCedula (filtro: s.clienteCedula = parámetro)
- TransportePasajeros tp — tp.idServicio (join con s.idServicio) para obtener tarifaPasajeros
- TransporteMercancias tme — tme.idServicio (join con s.idServicio) para tarifaMercancias
- EntregaDomicilio ed — ed.idServicio (join con s.idServicio) para tarifaDomicilio

JOINs usados: LEFT JOIN a las tablas de subtipo.

Justificación: no todos los servicios tienen subtipo; con LEFT JOIN obtenemos datos del servicio incluso si no existe la fila en la tabla de subtipo (evita filtrar servicios que no tengan subtipo).

SQL (adaptado al esquema):

```
SELECT s.idServicio, s.tipo, s.distancia, s.tarifa, s.horaInicio, s.horaFin, s.conductorCedula, s.placa,
    tp.tarifaPasajeros, tme.tarifaMercancias, ed.tarifaDomicilio
FROM Servicio s
LEFT JOIN TransportePasajeros tp ON tp.idServicio = s.idServicio
LEFT JOIN TransporteMercancias tme ON tme.idServicio = s.idServicio
LEFT JOIN EntregaDomicilio ed ON ed.idServicio = s.idServicio
WHERE s.clienteCedula = :p_cedula
ORDER BY s.horaInicio DESC;
```


All rows fetched: 1 in 0.271 seconds

	IDSERVICIO	TIPO	DISTANCIA	TARIFA	HORAINICIO	HORAFIN	CONDUCTORCEDULA
1	1	Domicilio	2	3	30/09/25 16:19:46.662	01/10/25 16:19:46.662	Cedula1

PLACA	TARIFAPASAJEROS	TARIFAMERCANCIAS	TARIFADOMICILIO
PLACA0001	803.07	15	1

RFC2 — Top 20 conductores que más servicios han prestado
Objetivo: listar los 20 conductores con mayor número de servicios prestados.
Tablas usadas:

- Servicio s — s.conductorCedula (contar servicios)
- Usuario u — u.cedula, u.nombre (para mostrar datos del conductor)

JOIN: INNER JOIN entre Servicio y Usuario (ON u.cedula = s.conductorCedula).
Justificación: solo interesan conductores que aparecen en Servicio (han prestado al menos un servicio). Si se quisiera incluir conductores con cero servicios, se usaría LEFT JOIN desde Usuario.
SQL:

```
SELECT u.cedula, u.nombre, COUNT(*) AS servicios_prestados
FROM Servicio s
JOIN Usuario u ON u.cedula = s.conductorCedula
WHERE s.conductorCedula IS NOT NULL
GROUP BY u.cedula, u.nombre
ORDER BY servicios_prestados DESC
FETCH FIRST 20 ROWS ONLY;
```

	CEDULA	NOMBRE	SERVICIOS_PRESTADOS
1	Cedula5	Usuario 5	1
2	Cedula31	Usuario 31	1
3	Cedula21	Usuario 21	1
4	Cedula34	Usuario 34	1
5	Cedula38	Usuario 38	1
6	Cedula39	Usuario 39	1
7	Cedula2	Usuario 2	1
8	Cedula4	Usuario 4	1
9	Cedula7	Usuario 7	1
10	Cedula11	Usuario 11	1
11	Cedula15	Usuario 15	1
12	Cedula27	Usuario 27	1
13	Cedula37	Usuario 37	1
14	Cedula41	Usuario 41	1
15	Cedula47	Usuario 47	1
16	Cedula48	Usuario 48	1
17	Cedula24	Usuario 24	1
18	Cedula25	Usuario 25	1
19	Cedula26	Usuario 26	1
20	Cedula10	Usuario 10	1

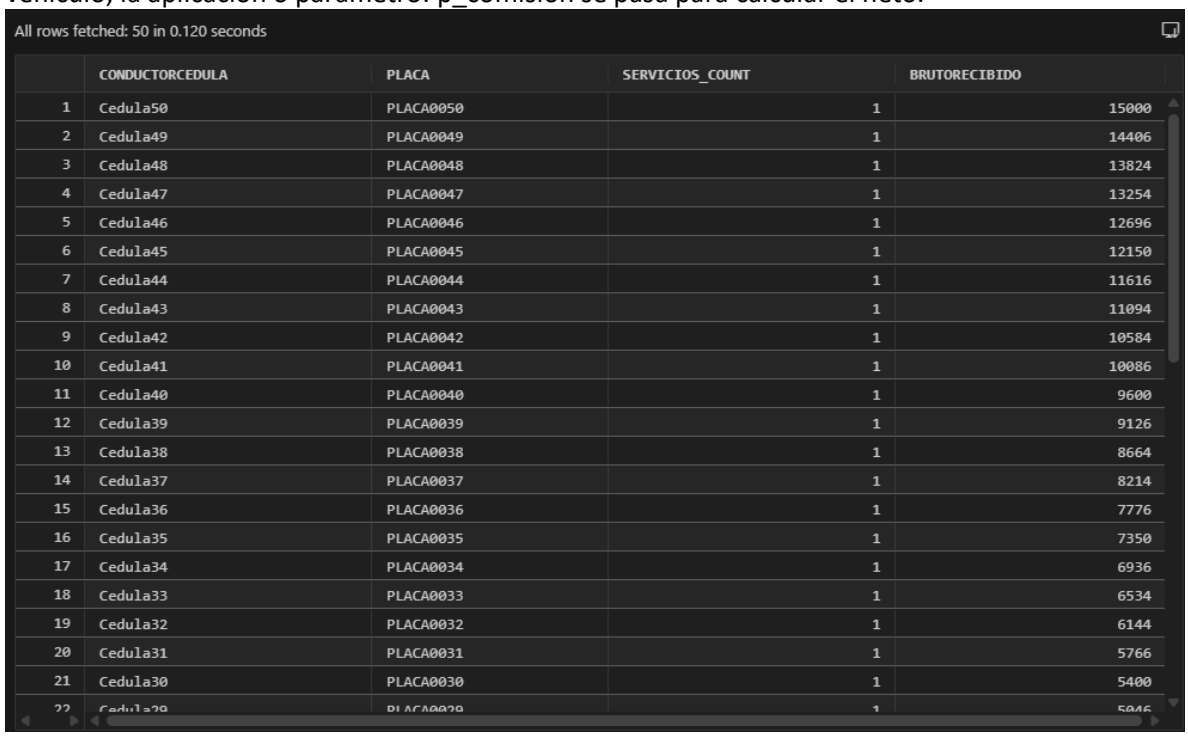
RFC3 — Total dinero obtenido por conductores por vehículo (después de comisión)
Objetivo: para cada conductor y cada vehículo sumar los valorServicio y aplicar una comisión.
Tablas usadas:

- Servicio s — s.conductorCedula, s.placa, s.valorServicio, s.estado (filtrar FINALIZADO)

JOINs: no son estrictamente necesarios si solo se reporta por s.conductorCedula y s.placa. Se puede JOIN con Usuario o Vehiculo si se necesita información adicional.
SQL:

```
SELECT s.conductorCedula, s.placa, COUNT(*) AS servicios_count,
    SUM(NVL(s.valorServicio,0)) AS brutoRecibido,
    SUM(NVL(s.valorServicio,0)) * (1 - :p_comision) AS netoDespuesComision
FROM Servicio s
WHERE s.estado = 'FINALIZADO' AND s.conductorCedula IS NOT NULL
GROUP BY s.conductorCedula, s.placa
```

ORDER BY netoDespuesComision DESC;
Justificación de agregaciones: agrupar por conductor y vehículo permite ver la recaudación por vehículo; la aplicación o parámetro: p_comision se pasa para calcular el neto.



All rows fetched: 50 in 0.120 seconds

	CONDUCTORCEDULA	PLACA	SERVICIOS_COUNT	BRUTORECIBIDO
1	Cedula50	PLACA0050	1	15000
2	Cedula49	PLACA0049	1	14406
3	Cedula48	PLACA0048	1	13824
4	Cedula47	PLACA0047	1	13254
5	Cedula46	PLACA0046	1	12696
6	Cedula45	PLACA0045	1	12150
7	Cedula44	PLACA0044	1	11616
8	Cedula43	PLACA0043	1	11094
9	Cedula42	PLACA0042	1	10584
10	Cedula41	PLACA0041	1	10086
11	Cedula40	PLACA0040	1	9600
12	Cedula39	PLACA0039	1	9126
13	Cedula38	PLACA0038	1	8664
14	Cedula37	PLACA0037	1	8214
15	Cedula36	PLACA0036	1	7776
16	Cedula35	PLACA0035	1	7350
17	Cedula34	PLACA0034	1	6936
18	Cedula33	PLACA0033	1	6534
19	Cedula32	PLACA0032	1	6144
20	Cedula31	PLACA0031	1	5766
21	Cedula30	PLACA0030	1	5400
22	Cedula29	PLACA0029	1	5046

RFC4 — Uso de servicios en una ciudad en un rango de fechas
Objetivo: mostrar la distribución de servicios y niveles por ciudad y rango de fechas (porcentaje y conteo), ordenada de mayor a menor uso.
Tablas usadas y joins:

- Servicio s — filtro por horaInicio entre fechas
- ServicioPunto sp — relaciona servicio ↔ punto
- PuntoGeografico pg — contiene ciudad (o ciudad_id) para identificar la ciudad asociada
- TransportePasajeros tp — para obtener nivelTransporte cuando aplique

JOINs: JOIN ServicioPunto + JOIN PuntoGeografico para identificar los servicios asociados a la ciudad. LEFT JOIN TransportePasajeros para traer nivelTransporte si existe.
Consideraciones: para evitar duplicar el conteo cuando un mismo servicio tenga múltiples puntos en la misma ciudad, la subconsulta usa SELECT DISTINCT s.idServicio.
SQL:

```
WITH servicios_ciudad AS (
  SELECT DISTINCT s.idServicio, s.tipo, tp.nivelTransporte
  FROM Servicio s
  JOIN ServicioPunto sp ON sp.idServicio = s.idServicio
  JOIN PuntoGeografico pg ON pg.idPunto = sp.idPunto
  LEFT JOIN TransportePasajeros tp ON tp.idServicio = s.idServicio
  WHERE pg.ciudad = :p_ciudad_id
    AND TRUNC(s.horaInicio) BETWEEN :p_fecha_inicio AND :p_fecha_fin
)
SELECT tipo,
      NVL(nivelTransporte,'N/A') AS nivelTransporte,
```

```
    COUNT(*) AS cantidad_servicios,
    ROUND( COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2 ) AS porcentaje_del_total
FROM servicios_ciudad
GROUP BY tipo, nivelTransporte
ORDER BY cantidad_servicios DESC;
```

All rows fetched: 1 in 0.189 seconds

	TIPO	NIVELTRANSPORTE	CANTIDAD_SERVICIOS	PORCENTAJE_DEL_TOTAL
1	Domicilio	Confort	1	100

5. Ejecución y pruebas (trazas sugeridas para la entrega)

Para comprobar que el esquema y la población cumplen los requisitos, sugerimos ejecutar las siguientes comprobaciones y anexar pantallazos en la entrega:

1. **Conteos básicos:**
    - SELECT COUNT(*) FROM Usuario WHERE tipo = 'CONDUCTOR'; → >= 100
    - SELECT COUNT(*) FROM Usuario WHERE tipo = 'CLIENTE'; → >= 200
    - SELECT COUNT(*) FROM Servicio; → >= (200 * 5) para cumplir promedio de 5 viajes por pasajero
2. **Prueba RFC1:** Ejecutar la consulta RFC1 para Cedula1 y verificar que devuelve las filas esperadas (captura de pantalla de resultados).
3. **Prueba RFC2 y RFC3:** Ejecutar y verificar que devuelven listas ordenadas y valores numéricos plausibles.
4. **Prueba RFC4:** Elegir ciudad = 1 y un rango de fechas que cubra las inserciones; comprobar que el total de cantidad_servicios coincide con el conteo de servicios únicos en la ciudad.
5. **Prueba de integridad:** Intentar insertar una Disponibilidad que se solape para la misma placa y verificar que la lógica/trigger o la comprobación en la aplicación lo impide (si implementaste trigger incluir trazas). Si aún no hay trigger, documentar la validación manual usada.

Estructura y responsabilidad de los paquetes principales

- **com.alpescab.model**
  Contiene las entidades JPA que representan las tablas del modelo relacional: Usuario, Vehiculo, Servicio, Disponibilidad, PuntoGeografico, Ciudad, TarjetaCredito, TransportePasajeros, TransporteMercancias, EntregaDomicilio, Rating, ServicioPunto, AlpesCab, etc. Cada entidad tiene las anotaciones JPA (@Entity, @Id, @ManyToOne, etc.) que reflejan las FKs y relaciones del modelo relacional.
- **com.alpescab.repository**
  Contiene interfaces que extienden JpaRepository (por ejemplo UsuarioRepository, VehiculoRepository, ServicioRepository, CityRepository, etc.). Proporcionan métodos CRUD automáticos y permiten consultas personalizadas cuando se necesite.
- **com.alpescab.service**
  La lógica de negocio intermedia entre controllers y repositorios. Aquí se implementan validaciones (p. ej. chequeo de solapamiento de disponibilidades), cálculo de niveles (estándar/confort/large) y orquestación de operaciones multi-tabla (registro de usuario + rol, asignación de conductor, etc.).

- **com.alpescab.controller**
  Expone endpoints REST (para Postman) y controladores MVC/Thymeleaf para páginas HTML. Ejemplos:
  - REST: /api/cities/register, /api/users/register-service-user, /api/users/register-conductor-user, /api/users/register-vehicle, /api/users/register-disponibilidad.
  - MVC: controladores que devuelven vistas Thymeleaf (index, cities, users, vehicles, servicios, etc.) para navegación web.
- **src/main/resources/templates**
  Plantillas Thymeleaf (HTML) que forman la interfaz mínima: index.html (home), vistas listadas (cities.html, users.html, vehicles.html, etc.) y formularios para crear/editar entidades.
- **application.properties**
  Archivo de configuración: datos de conexión a Oracle (spring.datasource.url, username, password), server.port, context-path, dialecto JPA. Para desarrollo existe la opción de usar un profile con H2.
- **pom.xml**
  Define dependencias (Spring Boot starter web, JPA, Thymeleaf, driver Oracle ojdbc8, H2 para dev, etc.), plugin de Spring Boot y configuración de compilación Java 17. Al ejecutar mvn spring-boot:run el plugin empaqueta y arranca la app.

Implementación de la lógica y mapeos importantes

- Las entidades reflejan la conversión UML → relacional: cada clase pasa a tabla; atributos simples a columnas; asociaciones 1:N o N:1 traducidas a FKs (por ejemplo Servicio.placa → FK a Vehiculo.placa); entidades especializadas (TransportePasajeros, TransporteMercancias, EntregaDomicilio) implementadas como tablas con idServicio PK que referencia Servicio (pattern table-per-type).
- Se añadieron checks e índices útiles: CHECK en Disponibilidad (horaFin > horaInicio), constraints FK consistentes, atributos UNIQUE donde aplica (ciudad.nombre, punto.direccion).
- Cálculo de tarifa: Servicio.valorServicio se definió como columna virtual (o calculada a nivel aplicación) distancia * tarifa. Para casos con tarifas específicas por tipo (pasajeros/mercancías/domicilio) hay tablas especializadas que almacenan tarifaPasajeros, tarifaMercancias, tarifaDomicilio.

Usuario Oracle para verificar tablas: ISIS2304F23202520

Vista de lo que se alcanzó realizar de la página web de Alpesca:

Alpescab

Ciudades Usuarios Vehículos Disponibilidades Servicios Entregas a Domicilio