Escuela Politécnica
Superior de Ingeniería
Universidad de La Laguna

Sistema de Gestión de Restauración de Land Rovers Clásicos

“Heritage Rover Works”

Alejandro Rodriguez Rojas - alu0101317038@ull.edu.es
Javier Francisco Díaz - alu0101049021@ull.edu.es
Aitor Manuel Perdomo Hermida - alu0101756424@ull.edu.es

Repositorio:
https://github.com/alerguezrojas/ADBD-PROYECTO-FINAL-LAND-ROVER.git

1. Resumen Ejecutivo

Este proyecto final de Administración y Diseño de Bases de Datos se centró en el diseño e implementación de un sistema de información robusto para gestionar vehículos y operaciones asociadas (inspecciones, mantenimiento, ventas, y gestión de personal) dentro de un contexto similar a Land Rover.

El proyecto genera un diseño conceptual completo que incorporó modelos avanzados (entidades débiles, jerarquías IS_A y relaciones complejas) y un modelo relacional normalizado y detallado. La implementación en PostgreSQL fue exitosa, utilizando lógica avanzada como disparadores (triggers) para hacer cumplir restricciones críticas (ej. evitar el solapamiento de inspecciones y mantenimientos) y garantizar la integridad referencial.

Además, se desarrolló una capa de aplicación web básica en Flask para demostrar la interacción y operatividad del sistema de base de datos. Los resultados confirman un sistema bien diseñado, con alta integridad de datos y listo para escalabilidad y futuras expansiones funcionales.

2. Objetivos del Proyecto

El proyecto tiene como objetivo principal diseñar e implementar una base de datos robusta y eficiente para modelar y gestionar la información.

Los objetivos específicos que satisface el sistema son:

- Gestión de Inventario y Compatibilidad: Controlar el stock de piezas y recambios, gestionando la complejidad de la compatibilidad cruzada entre diferentes modelos antiguos (relación M:N).
- Trazabilidad de Restauraciones: Documentar el ciclo de vida de los proyectos de restauración, desglosándolos por fases y asignando recursos humanos y materiales.
- Integridad de Servicios (Exclusión): Garantizar mediante reglas de negocio que un vehículo no pueda estar asignado simultáneamente a tareas incompatibles, como estar en proceso de restauración y ser alquilado para rodajes de cine.
- Gestión de Personal: Administrar roles diferenciados (Clientes y Empleados) y asegurar la supervisión adecuada de los proyectos (Inclusividad).
- Accesibilidad: Proveer una interfaz API REST para la gestión remota de la flota y el inventario.

3. Descripción del Contexto (Especificación de Requisitos)

La organización "Heritage Rover Works" requiere un sistema para digitalizar su flujo de trabajo. A continuación, se detallan las necesidades y cómo están justificadas las decisiones del modelado.

El sistema gestiona la información de una empresa de automoción clásica. A continuación, se detallan los requisitos de datos modelados:

Gestión de Personas (Jerarquía IS_A) Se define una entidad Persona que generaliza los datos comunes (DNI, nombre, contacto). Mediante una jerarquía total y disjunta, una persona debe ser clasificada como:

- Cliente: Se almacenan datos de facturación y fecha de alta.

- Empleado: Se almacenan datos laborales (NSS, salario, fecha de contratación).

Vehículos y Propiedad (1:N) Los clientes son propietarios de Vehículos. Un cliente puede poseer múltiples coches, pero un coche pertenece a un único titular actual. Se identifica por su número de bastidor (VIN).

Proyectos y Fases (Entidad Débil) La actividad principal es el Proyecto de restauración.

- Un vehículo puede tener varios proyectos históricos.

- Cada proyecto se compone de Fases (ej: Desmontaje, Pintura). La entidad Fase es débil respecto a Proyecto, ya que se identifica por un número secuencial que depende de la existencia del proyecto padre.

Logística y Proveedores (Relación Ternaria) Existe una relación ternaria Suministra entre Proveedor, Pieza y Proyecto. Esto permite registrar qué proveedor vendió una pieza específica para un proyecto concreto, guardando el precio de compra y la fecha de esa transacción.

Restricciones Avanzadas

- **Exclusión (Taller vs. Rodaje):** Un vehículo puede estar en una relación de Asiste_A (Rodajes de cine) o estar Sometido_A un Proyecto (Taller). El sistema impide mediante un disparador (trigger) que un vehículo participe en ambas relaciones simultáneamente si el proyecto está activo.

- **Inclusividad (Supervisión)**: Todo proyecto tiene un Supervisor (Empleado). El sistema requiere que el empleado que figura como supervisor esté incluido en el grupo de empleados que Trabajan_En ese proyecto.

4. Diseño Conceptual (Modelo Entidad-Relación)

Se definen las siguientes estructuras semánticas:

4.1 Entidades Principales

- Vehículo: Identificado por VIN (Vehicle Identification Number). Almacena Año, Matrícula, Color_Original y Tipo_Motor.
- Proyecto: Identificado por ID_Proyecto. Relaciona un vehículo con un rango temporal (Fecha_Inicio, Fecha_Fin) y un presupuesto.
- Pieza: Identificada por Ref_Pieza. Contiene la información técnica (Nombre, Descripción). Tiene una relación recursiva o de compatibilidad con Modelo_Catalogo.
- Proveedor: Identificado por CIF.

4.2. Relaciones y Cardinalidades

- Sometido_A (Vehículo - Proyecto): (0,n) - (1,1). Un vehículo puede tener varios proyectos en su histórico (o ninguno si está en stock), pero un proyecto pertenece a un único vehículo.
- Se_Divide_En (Proyecto - Fase): (1,1) - (1,n). Relación de dependencia fuerte. Un proyecto tiene al menos una fase.
- Suministra (Proveedor - Pieza): (0,n) - (0,n). Relación muchos a muchos con atributos (Precio).
- Trabaja_En (Empleado - Fase): (0,n) - (1,n). Un empleado trabaja en muchas fases, y una fase requiere al menos un empleado (o varios).

4.3. Supuestos Semánticos

- Jerarquía de Personas: Se ha optado por una jerarquía disjunta para simplificar la gestión de roles en la aplicación, asumiendo que un empleado no puede ser cliente en el mismo sistema.

- Eliminación de Vehículos: Se ha decidido que la eliminación de un vehículo implica la eliminación en cascada de su historial de proyectos y rodajes para mantener la integridad referencial.

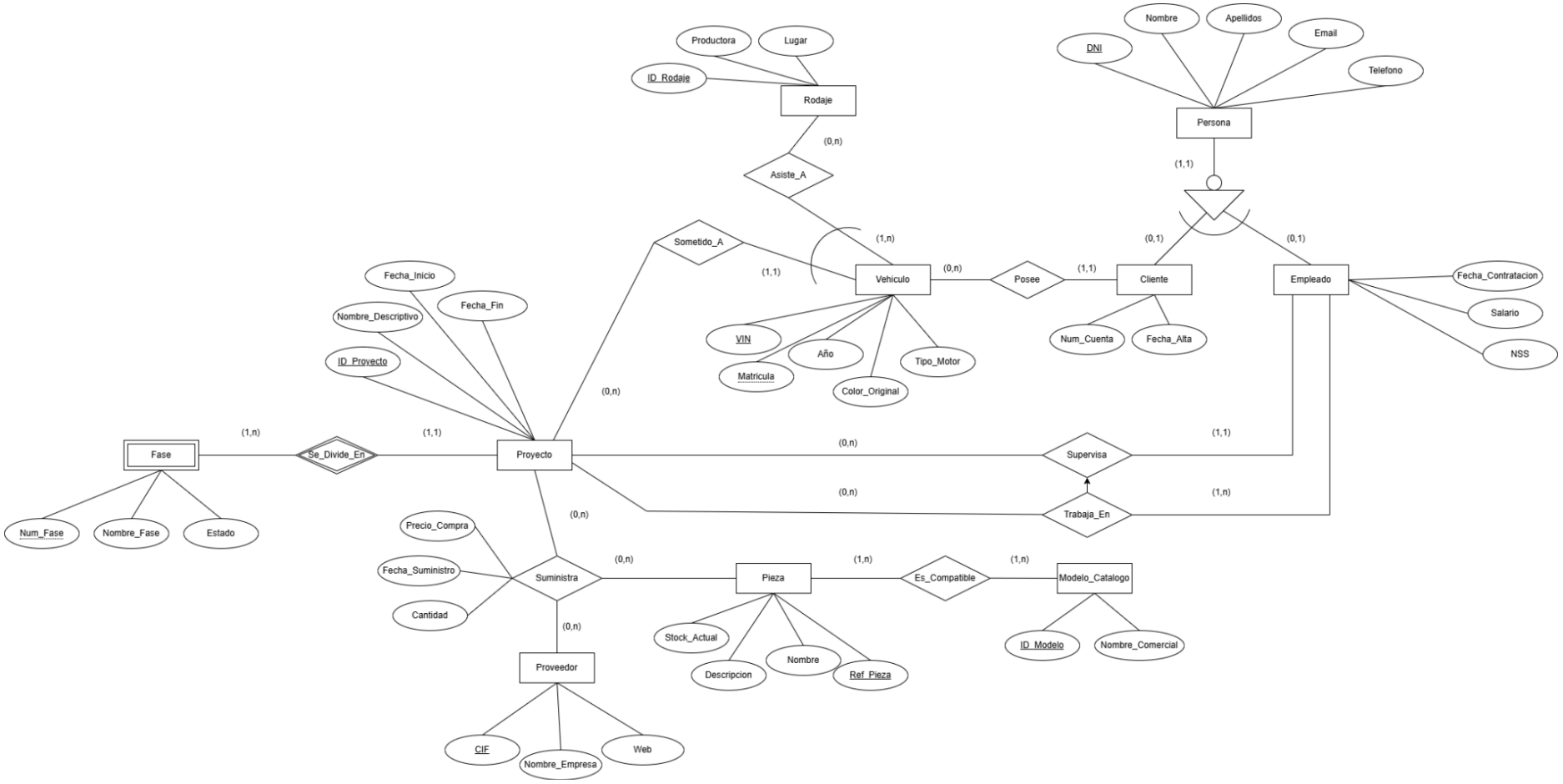
- Stock: El stock de piezas en la entidad Pieza es un campo calculado/actualizado automáticamente mediante triggers cada vez que se registra un suministro.

- Unicidad del VIN: Se asume que el VIN es estándar y único globalmente, sirviendo como identificador natural robusto.

- Secuencialidad de Fases: Las fases se numeran secuencialmente (1, 2, 3...) dentro de cada proyecto y no pueden solaparse temporalmente de forma ilógica (validado posteriormente vía Triggers).

- Histórico de Precios: El sistema almacena el precio actual de suministro. Si se requiere histórico de precios de compra, se debería consultar las facturas (fuera del alcance actual del E-R básico).

4.4. Diagrama de Entidad Relación



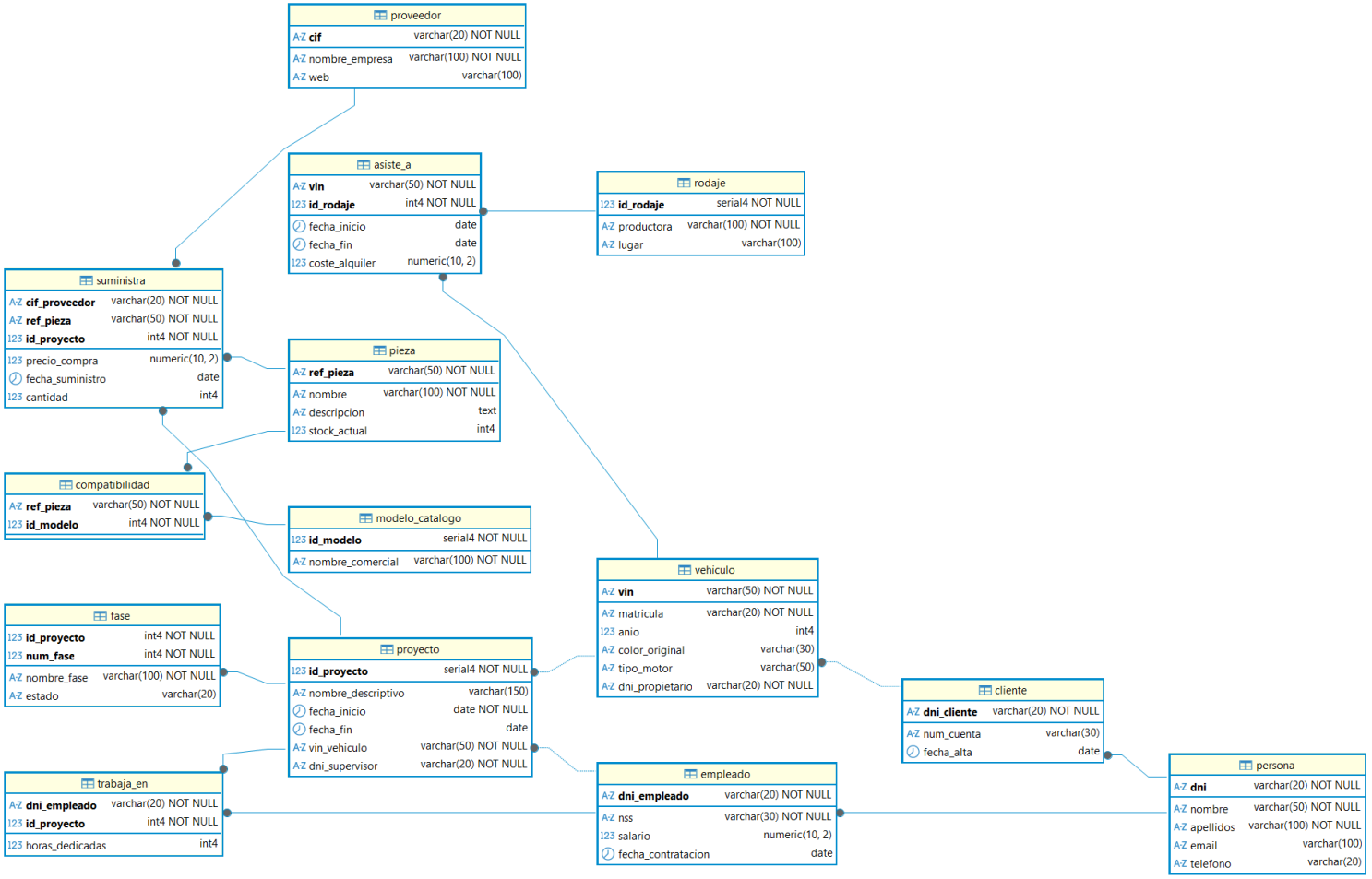
5. Diseño Lógico (Modelo Relacional)

A partir del diseño conceptual, se ha obtenido el siguiente esquema relacional normalizado:

Entidad	Atributo	Tipo de Llave	Restricciones/Notas
Persona	DNI	PK	Identificador unico (varchar)
	Nombre		
	Apellidos		
	Email		UNIQUE
	Telefono		
Cliente	DNI	PK, FK	Herencia de PERSONA
	Num_Cuenta		
	Fecha_Alta		
Empleado	DNI	PK,FK	Herencia de PERSONA
	NSS		UNIQUE
	Salario		
	Fecha_Concentracion		
Modelo_Catalogo	Id_Modelo	PK	
	Nombre_Comercial		
	Especificaciones		
Vehículo	VIN	PK	
	Matricula		UNIQUE
	Año		
	ID_Modelo	FK	Ref Modelo_CATALOGO
Proyecto	ID_Proyecto	PK, FK	

	Nun_Fase	PK	
	Descripcion		
	Estado		
	Coste_Estimado		
Pieza	Ref_Pieza	PK	
	Nombre		
	Stock_Actual		
Suministro	CIF_Proveedor	PK, FK	
	Ref_Pieza	PK, FK	
	Precio		
	Plazo_Entrega		
Asignación_Rodaje	ID_Rodaje	PK, FK	
	VIN	PK, FK	
	Fecha		
	Rol_Vehiculo		

5.1 Diagrama Relacional



6. Implementación en PostgreSQL

La implementación física de la base de datos se ha llevado a cabo sobre el Sistema Gestor de Bases de Datos (SGBD) PostgreSQL. Para garantizar la modularidad, el mantenimiento y la reproducibilidad del entorno, se ha optado por dividir el código SQL en scripts independientes según su funcionalidad.

A continuación, se detalla la estructura de implementación utilizada:

6.1. Definición del Esquema (schema.sql)

Este fichero contiene todas las sentencias DDL (Data Definition Language) necesarias para construir la estructura estática de la base de datos. Se ha realizado una traducción directa del modelo relacional obtenido en la fase de diseño.

Las características principales de esta implementación incluyen:

- **Creación de Tablas**: Se han definido las 14 tablas necesarias (PERSONA, VEHICULO, PROYECTO, etc.) respetando los tipos de datos más adecuados (VARCHAR, DATE, DECIMAL, SERIAL para autoincrementales).

- **Integridad Referencial**: Se han establecido todas las claves primarias (PK) y claves foráneas (FK) para vincular las tablas.

- **Jerarquía IS_A:** La generalización de Persona se ha implementado mediante la técnica de "Tabla por cada entidad", donde las tablas hijas CLIENTE y EMPLEADO referencian a la tabla padre PERSONA mediante una relación 1:1, garantizando la integridad de los datos comunes.

- **Entidades Débiles:** La tabla FASE se ha configurado con una clave primaria compuesta (ID_Proyecto, Num_Fase) y una restricción ON DELETE CASCADE en la clave foránea, asegurando que al eliminar un proyecto, sus fases desaparezcan automáticamente.

6.2. Lógica de Negocio y Restricciones Avanzadas (logic.sql)

Dado que el modelo estándar SQL no permite definir gráficamente ciertas restricciones complejas (como la exclusión entre relaciones o la actualización automática de campos calculados), se ha desarrollado un script específico en PL/pgSQL.

Este fichero implementa la "inteligencia" de la base de datos a través de Disparadores (Triggers) y Funciones Almacenadas:

1. **Automatización de Inventario (Trigger TRG_Actualizar_Stock):**

- Objetivo: Mantener la consistencia del stock en tiempo real.

- Funcionamiento: Se ha programado un trigger AFTER INSERT sobre la tabla ternaria SUMINISTRA. Cada vez que se registra una compra de material a un proveedor, el sistema captura la cantidad adquirida y actualiza automáticamente el campo Stock_Actual en la tabla maestra PIEZA.

2. **Integridad de Servicios (Trigger TRG_Check_Exclusion_Rodaje):**

- Objetivo: Cumplir la restricción de Exclusión definida en los requisitos (Un vehículo en taller no puede ir a rodaje).

- Funcionamiento: Se ha implementado un trigger BEFORE INSERT en la tabla ASISTE_A. Antes de asignar un vehículo a un rodaje, el sistema consulta si dicho vehículo tiene algún registro abierto (sin fecha de fin) en la tabla PROYECTO. Si es así, se lanza una excepción (RAISE EXCEPTION) que aborta la transacción e impide la inconsistencia de datos.

7. Carga de Datos

Para validar el correcto funcionamiento del diseño y de la lógica implementada, se ha procedido a poblar la base de datos y someterla a una batería de pruebas funcionales.

7.1. Poblado de la Base de Datos (data.sql)

Este fichero contiene las sentencias DML (Data Manipulation Language) para la carga masiva de datos. Se han insertado registros realistas para simular un entorno de producción verosímil.

El script sigue una estrategia de carga secuencial estricta para respetar la integridad referencial:

1. **Limpieza:** Ejecución de TRUNCATE ... CASCADE para garantizar un estado inicial limpio.

2. **Tablas Maestras:** Inserción de catálogos (MODELO, PIEZA, PROVEEDOR, RODAJE) y la superclase PERSONA.

3. **Tablas Dependientes:** Inserción de CLIENTE, EMPLEADO y VEHICULO.

4. **Tablas Transaccionales:** Generación de PROYECTO, FASE, y las relaciones complejas (SUMINISTRA, TRABAJA_EN, COMPATIBILIDAD).

8. Consultas de prueba (queries.sql y tests.sql)

Se han diseñado unos scripts específicos (queries.sql y test.sql) que actúan como banco de pruebas para verificar la capacidad de extracción de información del sistema y la corrección de las relaciones establecidas. Estos ficheros contienen consultas SQL que ponen a prueba el modelo relacional.

Se han validado los siguientes escenarios (queries.sql):

Prueba 1: Informe de Proyectos (Cruce Múltiple / JOINs): Esta consulta verifica la integridad de la jerarquía y las relaciones 1:N. Reconstruye la información completa de un proyecto uniendo cuatro tablas (PROYECTO, VEHICULO, CLIENTE, EMPLEADO) para generar un informe legible que muestra quién es el cliente, qué coche se está reparando y quién es el supervisor asignado.

sql > queries.sql

```sql
-- CONSULTAS DE PRUEBA - PROYECTO HERITAGE ROVER WORKS

-- 1. Listado detallado de proyectos (JOIN entre 4 tablas)
SELECT
    C.Nombre || ' ' || C.Apellidos AS Cliente,
    V.Matricula,
    P.Nombre_Descriptivo AS Proyecto,
    E.Nombre || ' ' || E.Apellidos AS Supervisor
FROM PROYECTO P
JOIN VEHICULO V ON P.VIN_Vehiculo = V.VIN
JOIN CLIENTE CL ON V.DNI_Propietario = CL.DNI_Cliente
JOIN PERSONA C ON CL.DNI_Cliente = C.DNI
JOIN EMPLEADO EMP ON P.DNI_Supervisor = EMP.DNI_Empleado
JOIN PERSONA E ON EMP.DNI_Empleado = E.DNI;
```

```
usuario@ubuntu:~/ADBD-PROYECTO-FINAL-LAND-ROVER$ sudo -u postgres psql -d landrovers -f sql/queries.sql
     cliente     | matricula |           proyecto           | supervisor
-----------------+-----------+------------------------------+------------
 Carlos Sainz    | M-1234-OP | Restauración Motor y Pintura | Laia Sanz
 Fernando Alonso | B-4321-KZ | Revisión Sistema Frenos      | Marc Coma
(2 rows)
```

Prueba 2: Análisis de Costes (Agregación en Relación Ternaria) Se ha testeado la relación ternaria SUMINISTRA mediante funciones de agregación (SUM, COUNT) y agrupación (GROUP BY). Esta consulta calcula el coste total de materiales invertidos en cada proyecto activo, validando que el sistema es capaz de trazar los suministros específicos asignados a una restauración concreta.

```sql
-- 2. Coste total de piezas por Proyecto (Agregación SUM y GROUP BY)
SELECT
    P.Nombre_Descriptivo,
    COUNT(S.Ref_Pieza) AS Num_Piezas_Compradas,
    COALESCE(SUM(S.Precio_Compra * S.Cantidad), 0) AS Coste_Total_Piezas
FROM PROYECTO P
LEFT JOIN SUMINISTRA S ON P.ID_Proyecto = S.ID_Proyecto
GROUP BY P.ID_Proyecto, P.Nombre_Descriptivo;
```

```
      nombre_descriptivo      | num_piezas_compradas | coste_total_piezas
------------------------------+----------------------+--------------------
 Revisión Sistema Frenos      |                    1 |              25.00
 Restauración Motor y Pintura |                    1 |             150.00
(2 rows)
```

Prueba 3: Búsqueda de Compatibilidad (Relación M:N) Esta consulta valida la relación de "Muchos a Muchos" entre PIEZA y MODELO. Simula un caso de uso real de taller: buscar qué recambios en stock son compatibles con un modelo de vehículo específico (por ejemplo, "Defender"). Para ello, la consulta atraviesa la tabla intermedia COMPATIBILIDAD, demostrando la correcta normalización del catálogo de piezas.

```sql
-- 3. Buscar piezas compatibles con Defender (Relación M:N)
SELECT
    M.Nombre_Comercial AS Modelo,
    P.Nombre AS Pieza_Compatible,
    P.Stock_Actual
FROM MODELO_CATALOGO M
JOIN COMPATIBILIDAD C ON M.ID_Modelo = C.ID_Modelo
JOIN PIEZA P ON C.Ref_Pieza = P.Ref_Pieza
WHERE M.Nombre_Comercial LIKE '%Defender%';
```

```
      modelo       | pieza_compatible | stock_actual
-------------------+------------------+--------------
 Defender 90 Tdi   | Filtro Aceite    |           52
 Defender 110 V8   | Filtro Aceite    |           52
(2 rows)
```

Por otra parte, para garantizar la robustez del sistema, se ha diseñado y ejecutado un guión de pruebas integral (tests.sql) que abarca las 14 tablas del sistema:

Prueba 1: Gestión de Jerarquía y Roles

Objetivo: Verificar la integridad referencial en la relación de herencia entre PERSONA y CLIENTE.

Código Ejecutado:

```sql
-- ------------------------------------------------------------------------------
-- BLOQUE 1: GESTIÓN DE PERSONAS Y ROLES (Tablas: PERSONA, CLIENTE, EMPLEADO)
-- Objetivo: Verificar la jerarquía IS_A, la inserción en tablas vinculadas y la integridad.
-- ------------------------------------------------------------------------------

-- 1.1. Inserción de una nueva Persona en la superclase.
INSERT INTO PERSONA (DNI, Nombre, Apellidos, Email, Telefono)
VALUES ('99999999Z', 'Test Persona', 'Apellido Test', 'test@mail.com', '600000000');

-- 1.2. Especialización: Registro de la entidad dependiente CLIENTE.
INSERT INTO CLIENTE (DNI_Cliente, Num_Cuenta, Fecha_Alta)
VALUES ('99999999Z', 'ES0000000000000000000', CURRENT_DATE);

-- 1.3. Verificación de la integridad de los datos mediante reunión natural (JOIN).
SELECT P.Nombre, P.Apellidos, C.Num_Cuenta
FROM PERSONA P
JOIN CLIENTE C ON P.DNI = C.DNI_Cliente
WHERE P.DNI = '99999999Z';

-- 1.4. REVERSIÓN (Limpieza): Eliminación de los registros de prueba.
DELETE FROM CLIENTE WHERE DNI_Cliente = '99999999Z';
DELETE FROM PERSONA WHERE DNI = '99999999Z';
```

Salida Obtenida: La consulta devuelve una tupla combinada, confirmando que los datos comunes y específicos están correctamente vinculados

Prueba 2: Operaciones en Catálogos Maestros

Objetivo: Validar la persistencia de datos en tablas auxiliares (MODELO, PROVEEDOR, RODAJE).

Código Ejecutado:

```sql
-- ------------------------------------------------------------------------------
-- BLOQUE 2: CATÁLOGOS AUXILIARES (Tablas: MODELO_CATALOGO, PROVEEDOR, RODAJE)
-- Objetivo: Validación de operaciones CRUD en tablas maestras sin dependencias fuertes.
-- ------------------------------------------------------------------------------

-- 2.1. Inserción de un nuevo registro en el catálogo de modelos.
INSERT INTO MODELO_CATALOGO (Nombre_Comercial) VALUES ('Land Rover Series I 86" (Versión Test)');

-- 2.2. Modificación (UPDATE) de un atributo en la tabla de proveedores.
UPDATE PROVEEDOR SET Web = 'www.britpart-updated-test.com' WHERE Nombre_Empresa LIKE 'Britpart%';

-- 2.3. Registro de un nuevo evento de Rodaje.
INSERT INTO RODAJE (Productora, Lugar) VALUES ('Test Productions', 'Desierto de Tabernas');

-- 2.4. Consulta de verificación de las operaciones DML realizadas.
SELECT * FROM MODELO_CATALOGO WHERE Nombre_Comercial LIKE '%(Versión Test)';
SELECT Nombre_Empresa, Web FROM PROVEEDOR WHERE Nombre_Empresa LIKE 'Britpart%';

-- 2.5. REVERSIÓN (Limpieza):
DELETE FROM MODELO_CATALOGO WHERE Nombre_Comercial LIKE '%(Versión Test)';
UPDATE PROVEEDOR SET Web = 'www.britpart.com' WHERE Nombre_Empresa LIKE 'Britpart%';
DELETE FROM RODAJE WHERE Productora = 'Test Productions';
```

Salida Obtenida: Se observa el nuevo registro en el catálogo de modelos y la URL modificada en la tabla de proveedores.

Prueba 3: Actualización de Vehículos

Objetivo: Verificar la modificación de atributos en la entidad VEHICULO.

Código Ejecutado:

```sql
-- ------------------------------------------------------------------------------
-- BLOQUE 3: GESTIÓN DE VEHÍCULOS (Tabla: VEHICULO)
-- Objetivo: Verificación de persistencia en actualizaciones de atributos.
-- ------------------------------------------------------------------------------

-- 3.1. Consulta del estado inicial (Color actual: 'Verde Inglés').
SELECT VIN, Matricula, Color_Original FROM VEHICULO WHERE Matricula = 'M-1234-OP';

-- 3.2. Actualización: Modificación del color del vehículo a 'Rojo Fuego'.
UPDATE VEHICULO SET Color_Original = 'Rojo Fuego' WHERE Matricula = 'M-1234-OP';

-- 3.3. Verificación del cambio de estado en la base de datos.
SELECT VIN, Matricula, Color_Original FROM VEHICULO WHERE Matricula = 'M-1234-OP';

-- 3.4. REVERSIÓN (Limpieza): Restauración del valor original.
UPDATE VEHICULO SET Color_Original = 'Verde Inglés' WHERE Matricula = 'M-1234-OP';
```

Salida Obtenida: El campo Color_Original refleja el cambio de 'Verde Inglés' a 'Rojo Fuego'.

Prueba 4: Relación Muchos a Muchos (Inventario)

Objetivo: Comprobar la correcta asociación entre PIEZA y MODELO a través de la tabla intermedia COMPATIBILIDAD.

Código Ejecutado:

```sql
-- ------------------------------------------------------------------------------
-- BLOQUE 4: INVENTARIO Y COMPATIBILIDAD (Tablas: PIEZA, COMPATIBILIDAD)
-- Objetivo: Gestión de relación M:N entre Piezas y Modelos.
-- ------------------------------------------------------------------------------

-- 4.1. Alta de una nueva referencia de pieza en almacén.
INSERT INTO PIEZA (Ref_Pieza, Nombre, Descripcion, Stock_Actual)
VALUES ('TEST-PART', 'Pieza Universal', 'Componente de prueba M:N', 10);

-- 4.2. Asignación de compatibilidad (M:N): Vinculación de la pieza con el Modelo ID 1.
INSERT INTO COMPATIBILIDAD (Ref_Pieza, ID_Modelo) VALUES ('TEST-PART', 1);

-- 4.3. Verificación de la integridad de la relación establecida.
SELECT P.Nombre as Pieza, M.Nombre_Comercial as Modelo
FROM PIEZA P
JOIN COMPATIBILIDAD C ON P.Ref_Pieza = C.Ref_Pieza
JOIN MODELO_CATALOGO M ON C.ID_Modelo = M.ID_Modelo
WHERE P.Ref_Pieza = 'TEST-PART';

-- 4.4. REVERSIÓN (Limpieza):
DELETE FROM COMPATIBILIDAD WHERE Ref_Pieza = 'TEST-PART';
DELETE FROM PIEZA WHERE Ref_Pieza = 'TEST-PART';
```

Salida Obtenida: La consulta muestra la pieza vinculada correctamente al modelo "Land Rover Series I"

Prueba 5: Validación de Restricciones (CHECK)

Objetivo: Demostrar que la base de datos rechaza datos inválidos en la tabla FASE.

Código Ejecutado:

```sql
-- ------------------------------------------------------------------------------
-- BLOQUE 5: PROYECTOS Y FASES (Tablas: PROYECTO, FASE, TRABAJA_EN)
-- Objetivo: Validación de restricciones CHECK y flujo de trabajo.
-- ------------------------------------------------------------------------------

-- 5.1. Apertura de un Proyecto temporal para pruebas.
-- VIN: 'SALLHV654321', Supervisor: '33333333C'.
INSERT INTO PROYECTO (Nombre_Descriptivo, Fecha_Inicio, VIN_Vehiculo, DNI_Supervisor)
VALUES ('Proyecto Test de Integridad', CURRENT_DATE, 'SALLHV654321', '33333333C');

-- 5.2. Asignación de recursos humanos (Tabla TRABAJA_EN).
-- Se asigna al propio supervisor para cumplir con la restricción de Inclusividad.
INSERT INTO TRABAJA_EN (DNI_Empleado, ID_Proyecto, Horas_Dedicadas)
VALUES ('33333333C', (SELECT MAX(ID_Proyecto) FROM PROYECTO), 5);

-- 5.3. Prueba de Restricción CHECK en la tabla FASE.
-- Intento de inserción de un estado no válido ('Dudoso') fuera del dominio permitido.
-- Se espera un error de violación de restricción check.
INSERT INTO FASE (ID_Proyecto, Num_Fase, Nombre_Fase, Estado)
VALUES ((SELECT MAX(ID_Proyecto) FROM PROYECTO), 1, 'Fase Test', 'Dudoso');

-- 5.4. REVERSIÓN (Limpieza):
-- Eliminación del proyecto de prueba y sus dependencias (Cascade delete en FASE).
DELETE FROM TRABAJA_EN WHERE ID_Proyecto = (SELECT MAX(ID_Proyecto) FROM PROYECTO WHERE Nombre_Descriptivo = 'Proyecto T
DELETE FROM PROYECTO WHERE Nombre_Descriptivo = 'Proyecto Test de Integridad';
```

Salida Obtenida: El sistema devuelve un error de violación de la restricción check (fase_estado_check), impidiendo la inserción de valores fuera del dominio permitido.

Prueba 6: Disparador de Actualización de Stock

Objetivo: Verificar que el trigger actualizar_stock_compra incrementa el inventario automáticamente tras una compra.

Código Ejecutado:

```sql
-- ------------------------------------------------------------------------------
-- BLOQUE 6: DISPARADOR DE STOCK (Tabla: SUMINISTRA)
-- Objetivo: Verificación de la lógica de negocio (Trigger) para actualización automática.
-- ------------------------------------------------------------------------------

-- 6.1. Consulta del stock inicial de la referencia 'GME123'.
SELECT Ref_Pieza, Stock_Actual FROM PIEZA WHERE Ref_Pieza = 'GME123';

-- 6.2. Simulación de compra: Inserción en la relación ternaria SUMINISTRA (+10 unidades).
INSERT INTO SUMINISTRA (CIF_Proveedor, Ref_Pieza, ID_Proyecto, Precio_Compra, Fecha_Suministro, Cantidad)
VALUES ('B12345678', 'GME123', 1, 15.00, CURRENT_DATE, 10);

-- 6.3. Verificación del stock final.
-- El valor debe haberse incrementado en 10 unidades automáticamente.
SELECT Ref_Pieza, Stock_Actual FROM PIEZA WHERE Ref_Pieza = 'GME123';

-- 6.4. REVERSIÓN (Limpieza):
-- Eliminación del registro de suministro.
DELETE FROM SUMINISTRA WHERE Ref_Pieza = 'GME123' AND Cantidad = 10 AND Fecha_Suministro = CURRENT_DATE;
-- Corrección manual del stock para restaurar el estado original.
UPDATE PIEZA SET Stock_Actual = Stock_Actual - 10 WHERE Ref_Pieza = 'GME123';
```

Salida Obtenida: Se evidencia que el stock ha aumentado en 10 unidades sin necesidad de una instrucción UPDATE explícita sobre la tabla PIEZA.

Prueba 7: Disparador de Exclusión (Integridad de Negocio)

Objetivo: Validar que el trigger validar_exclusion_rodaje impide asignar a un rodaje un vehículo que se encuentra en taller.

Código Ejecutado:

```sql
-- ------------------------------------------------------------------------------
-- BLOQUE 7: DISPARADOR DE EXCLUSIÓN (Tabla: ASISTE_A)
-- Objetivo: Validar la restricción de exclusividad entre Taller y Rodaje.
-- ------------------------------------------------------------------------------

-- 7.1. Confirmación de que el vehículo 'SALLHV123456' tiene un proyecto activo.
SELECT ID_Proyecto, VIN_Vehiculo, Fecha_Fin FROM PROYECTO WHERE VIN_Vehiculo = 'SALLHV123456';

-- 7.2. Intento de asignación a un Rodaje.
-- Se espera una excepción lanzada por el disparador impidiendo la operación.
INSERT INTO ASISTE_A (VIN, ID_Rodaje, Fecha_Inicio, Fecha_Fin, Coste_Alquiler)
VALUES ('SALLHV123456', 1, CURRENT_DATE, CURRENT_DATE + 5, 5000);

-- No requiere reversión ya que la operación es abortada por el sistema.
```

Salida Obtenida: La base de datos aborta la transacción y muestra el mensaje de error personalizado definido en la función PL/pgSQL: "ERROR DE EXCLUSIÓN: El vehículo... está en el taller".

9. Implementación de API REST y Aplicación Web

Para cumplir con los requisitos de explotación y accesibilidad, se ha desarrollado una API RESTful completa utilizando Python (Flask). Esta interfaz permite realizar operaciones CRUD sobre la base de datos PostgreSQL, respetando todas las reglas de negocio y restricciones de integridad definidas.

9.1. Arquitectura de la API

El sistema sigue el patrón MVC. Los controladores (rutas Flask) reciben peticiones HTTP, el modelo ejecuta sentencias SQL parametrizadas con psycopg2 (gestionando transacciones con commit/rollback) y la vista devuelve respuestas serializadas en JSON.

9.2. Especificación de Endpoints

A continuación, se detalla la tabla completa de operaciones disponibles. En la columna de petición se incluye el comando cURL completo listo para ser ejecutado en una terminal.

(i) Descripción y Endpoint	(ii) Método	(iii) Ejemplo de Petición (Comando cURL)	(iv) Ejemplo de Respuesta
Dashboard Stats /api/dashboard	GET	curl -X GET http://localhost:5000/api/dashboard	{"vehiculos": 5, "proyectos": 2}
Listar Personas /api/personas	GET	curl -X GET http://localhost:5000/api/personas	[{"DNI": "11A", "Rol": "Cliente"}]
Crear Persona /api/personas	POST	curl -X POST http://localhost:5000/api/personas -H "Content-Type: application/json" -d '{"DNI": "99Z", "Nombre": "Ana", "Rol": "Cliente", "Num_Cuenta": "ES21"}'	{"msg": "Cliente registrado"}
Borrar Persona /api/personas/<dni>	DELETE	curl -X DELETE http://localhost:5000/api/personas/99Z	{"msg": "Persona eliminada"}
Listar Vehículos /api/vehiculos	GET	curl -X GET http://localhost:5000/api/vehiculos	[{"VIN": "SA...", "Matricula": "TF-1"}]
Crear Vehículo /api/vehiculos	POST	curl -X POST http://localhost:5000/api/vehiculos -H "Content-Type: application/json" -d '{"VIN": "T01", "Matricula": "M-00", "Anio": 1980, "Color": "Verde", "Motor": "V8", "DNI": "11A"}'	{"msg": "Vehículo creado"}
Borrar Vehículo /api/vehiculos/<vin>	DELETE	curl -X DELETE http://localhost:5000/api/vehiculos/T01	{"msg": "Vehículo eliminado"}
Listar Proyectos /api/proyectos	GET	curl -X GET http://localhost:5000/api/proyectos	[{"ID": 1, "Nombre": "Pintura"}]
Abrir Proyecto /api/proyectos	POST	curl -X POST http://localhost:5000/api/proyectos -H "Content-Type: application/json" -d '{"Nombre": "Motor", "VIN": "T01", "Supervisor": "EMP01"}'	{"msg": "Proyecto abierto"}
Cancelar Proyecto /api/proyectos/<id>	DELETE	curl -X DELETE http://localhost:5000/api/proyectos/	{"msg": "Proyecto eliminado"}

		1	
Listar Catálogo /api/piezas	GET	curl -X GET http://localhost:5000/api/piezas	[{"Ref": "GME123", "Stock": 50}]
Crear Pieza /api/piezas	POST	curl -X POST http://localhost:5000/api/piezas -H "Content-Type: application/json" -d '{"Ref_Pieza": "FILTRO", "Nombre": "Filtro Aire", "Stock_Inicial": 0}'	{"msg": "Referencia creada"}
Borrar Pieza /api/piezas/<ref>	DELETE	curl -X DELETE http://localhost:5000/api/piezas/FILTRO	{"msg": "Pieza eliminada"}
Comprar (Trigger) /api/comprar_pieza	POST	curl -X POST http://localhost:5000/api/comprar_pieza -H "Content-Type: application/json" -d '{"Ref_Pieza": "GME123", "Cantidad": 10}'	{"msg": "Compra OK. Stock +10"}
Listar Rodajes /api/rodajes	GET	curl -X GET http://localhost:5000/api/rodajes	[{"VIN": "SA...", "Lugar": "Cine"}]
Enviar Rodaje (Trig)/api/enviar_rodaje	POST	curl -X POST http://localhost:5000/api/enviar_rodaje -H "Content-Type: application/json" -d '{"VIN": "T01", "Coste": 2000}'	{"msg": "Enviado a rodaje"}
Fin Rodaje /api/rodajes/<v>/<i>	DELETE	curl -X DELETE http://localhost:5000/api/rodajes/T01/1	{"msg": "Rodaje finalizado"}

9.3. Interfaz de Usuario (Frontend)

Para consumir esta API de forma visual, se ha desarrollado una interfaz web (index.html) utilizando HTML5, Tailwind CSS y Alpine.js.

La aplicación permite navegar por diferentes módulos (Dashboard, Flota, Proyectos, Almacén) y visualizar en tiempo real los efectos de los disparadores de base de datos, como el bloqueo por exclusión (mostrando alertas de error rojas) o la actualización automática de stock en las tablas visuales.

10. Presupuesto del Proyecto

El presente apartado detalla la estimación económica para el desarrollo del sistema "Heritage Rover Works". El cálculo se ha realizado en base al esfuerzo en horas invertido

por el equipo de desarrollo (3 integrantes), aplicando tarifas de mercado para perfiles de ingeniería junior.

10.1. Costes de Personal (Mano de Obra)

Se han contabilizado las horas dedicadas a cada fase del ciclo de vida del software, asignando roles específicos según la tarea realizada.

Fase del Proyecto	Rol Asignado	Horas Estimadas	Coste Hora (€)	Total Fase (€)
1. Análisis y Requisitos	Analista Funcional	12 h	35,00€	420,00€
2. Diseño de Base de Datos	Arquitecto de Datos	18 h	40,00€	720,00€
3. Implementación SQL	DBA (Admin. BBDD)	20 h	30,00€	600,00€
4. Desarrollo Backend (API)	Desarrollador Backend	25 h	30,00€	750,00€
5. Desarrollo Frontend (Web)	Desarrollador Frontend	15 h	30,00€	450,00€
6. Pruebas y Documentación	QA / Tester	10 h	25,00€	250,00€
SUBTOTAL PERSONAL		100 h		3.190,00€

10.2. Costes de Recursos (Software y Hardware)

Aunque se prioriza el software libre, se imputan costes de amortización de hardware y gastos operativos necesarios para el desarrollo.

- *Cálculo de Amortización:* Se estima el uso de 3 equipos portátiles de gama media-alta (valor medio 1.200€) durante 1 mes de desarrollo.
- *Gastos Generales:* Proporcional de electricidad y conexión a internet de fibra óptica para 3 puestos de trabajo remotos.

Recurso / Concepto	Detalle	Coste (€)
Software de Base	PostgreSQL, Python, VS Code (Open Source)	0,00€
Amortización Hardware	Desgaste de 3 equipos	300,00€

	informáticos	
Suministros y Comunicaciones	Electricidad e Internet (Fibra Óptica)	120,00€
Servidor de Pruebas (VPS)	Despliegue en la nube (Estimación 1 mes)	15,00€
Material de Oficina	Licencias menores y varios	50,00€
SUBTOTAL RECURSOS		485,00€

10.3. Resumen Económico

Concepto	Importe
Costes de Personal	3.190,00€
Costes de Recursos y Gastos	485,00€
TOTAL EJECUCIÓN MATERIAL	3.675,00€
Imprevistos (10%)	367,50€
PRESUPUESTO FINAL	4.042,50€

11. Conclusiones

El desarrollo del proyecto "Heritage Rover Works" ha permitido cumplir satisfactoriamente con todos los objetivos de la asignatura, obteniendo las siguientes conclusiones principales:

- Modelado Complejo: Hemos logrado integrar correctamente requisitos avanzados como relaciones ternarias, entidades débiles y jerarquías en un esquema relacional robusto y normalizado.

- Integridad de Datos: La implementación de Triggers (PL/pgSQL) ha sido clave. Al programar la lógica de negocio (control de stock y exclusión de rodajes) dentro de la base de datos, garantizamos que la información sea siempre consistente, independientemente de quién acceda a ella.

- Sistema Funcional: La creación de la API REST con Flask y la interfaz web demuestra que el diseño no es solo teórico, sino que sirve como base para una aplicación real y útil para el usuario final.

En resumen, este proyecto ha consolidado nuestros conocimientos en diseño y administración de bases de datos, permitiéndonos simular un entorno de trabajo profesional con herramientas estándar de la industria.