# REPORTE DE PRÁCTICA NO. 2

## NOMBRE DE LA PRÁCTICA: Flotilla

**ALUMNO:**

**Janett Hernández**
**Adán Villeda Trejo**

# 1. Introducción

La gestión eficiente de una flotilla vehicular requiere un control de distintos aspectos como el mantenimiento, documentación de cada unidad, los conductores asignados y las rutas recorridas. Para cumplir con este propósito, se diseñó una base de datos que integra las entidades principales relacionadas con la operación de una flotilla de autos.

El modelo propuesto incluye tablas para registrar mantenimientos, documentación vehicular, autos, conductores y rutas. Con la implementación de llaves primarias y foráneas, se asegura la gestión de los datos y se facilita la consulta de información relevante, como el historial de un vehículo, los documentos vigentes de cada unidad o las rutas asignadas a un conductor específico.

# Marco Teorico

## Procedimientos almacenados (Procedure)

Un procedimiento almacenado es un bloque de código PL/SQL o SQL que se almacena en la base de datos y puede ser ejecutado posteriormente mediante una llamada específica. Sirve para automatizar tareas recurrentes en la gestión de datos, aceptar parámetros de entrada y definir operaciones complejas que combinan varias instrucciones SQL. Los procedimientos pueden ser invocados desde aplicaciones externas o internamente desde otros procedimientos, facilitando la reutilización y centralización de la lógica de negocio.

## Funciones (Function)

Las funciones son similares a los procedimientos almacenados pero se distinguen por siempre retornar un valor como resultado de su ejecución. Se utilizan para realizar cálculos, transformar datos o validar información, y su valor de retorno puede ser empleado directamente en consultas SQL, expresiones o incluso dentro de otros procedimientos y funciones. Además, las funciones aceptan parámetros y pueden tener lógica condicional, repetitiva y operaciones complejas.

## Estructuras de control condicionales y repetitivas

Las estructuras de control condicionales permiten tomar decisiones en el flujo del programa, ejecutando diferentes bloques de código según el resultado de una condición lógica (por ejemplo, IF, CASE). Las estructuras repetitivas, por otro lado, facilitan la ejecución de instrucciones múltiples veces, ya sea un número fijo de iteraciones o mientras se cumpla una condición (por ejemplo, LOOP, WHILE, FOR). Estas estructuras son esenciales para procesar lotes de datos y controlar el flujo de ejecución en procedimientos y funciones.

## Disparadores (Triggers)

Los disparadores son bloques de código que se ejecutan automáticamente en respuesta a ciertos eventos en la base de datos, como inserciones, actualizaciones o eliminaciones en una tabla. Permiten automatizar tareas de auditoría, validación o actualización sin necesidad de intervención manual, asegurando la integridad y coherencia de los datos. Los triggers se definen para actuar antes o después de los eventos especificados, y pueden incluir lógica condicional y llamadas a procedimientos o funciones.

## Fragmentación Vertical

La fragmentación vertical se refiere a la división de una relación en subconjuntos de atributos (columna); cada subconjunto (fragmento) se guarda en un nodo diferente y cada fragmento tiene columnas únicas, con la excepción de la columna clave, la cual es común a todos los fragmentos. Esto es el equivalente de la sentencia SELECT columna1, columna2 INTO Nueva$_{T}ablaFROMTabla$.

## Procesos ETL

Los procesos ETL son una parte de la integración de datos, pero es un elemento importante cuya función completa el resultado de todo el desarrollo de la cohesión de aplicaciones y sistemas.
La palabra ETL corresponde a las siglas en inglés de:
- Extraer: extract.
- Transformar: transform.
- Y Cargar: load.

Con ello, queremos decir que todo proceso ETL consta precisamente de estas tres fases: extracción, transformación y carga. Vamos a definir en qué consisten cada una de estas fases.

## SELECT + INTO FILE

Es una sentencia SQL que exporta los resultados de una consulta SELECT a un archivo de texto en el servidor de base de datos, con un formato específico. Se utiliza para realizar operaciones de exportación de datos, como la creación de copias de seguridad, el análisis de datos o la transferencia de datos a otros sistemas.

## LOAD

En bases de datos, LOAD (o carga) se refiere al proceso de mover datos de un origen, como un archivo de texto, a una tabla de la base de datos de manera eficiente. Se utiliza para transferir grandes volúmenes de datos y puede sobrescribir, añadir o crear nuevas tablas. El comando SQL específico suele ser LOAD DATA, mientras que los sistemas de bases de datos también pueden tener utilidades independientes como dbload (IBM) o SQL Loader (Oracle).

## SELECT con tablas de dos bases de datos

Una consulta SELECT con tablas de dos bases de datos permite obtener datos de múltiples tablas, ya sea combinando sus resultados con UNION o relacionándolas con JOIN para combinar filas basándose en una columna común. Se debe especificar la tabla de origen para cada columna, usando el nombre de la tabla seguido de un punto (ej., tabla1.columna) o alias (ej., t1.columna), especialmente si hay nombres de columna duplicados.

## 2. Modelo Entidad - Relación

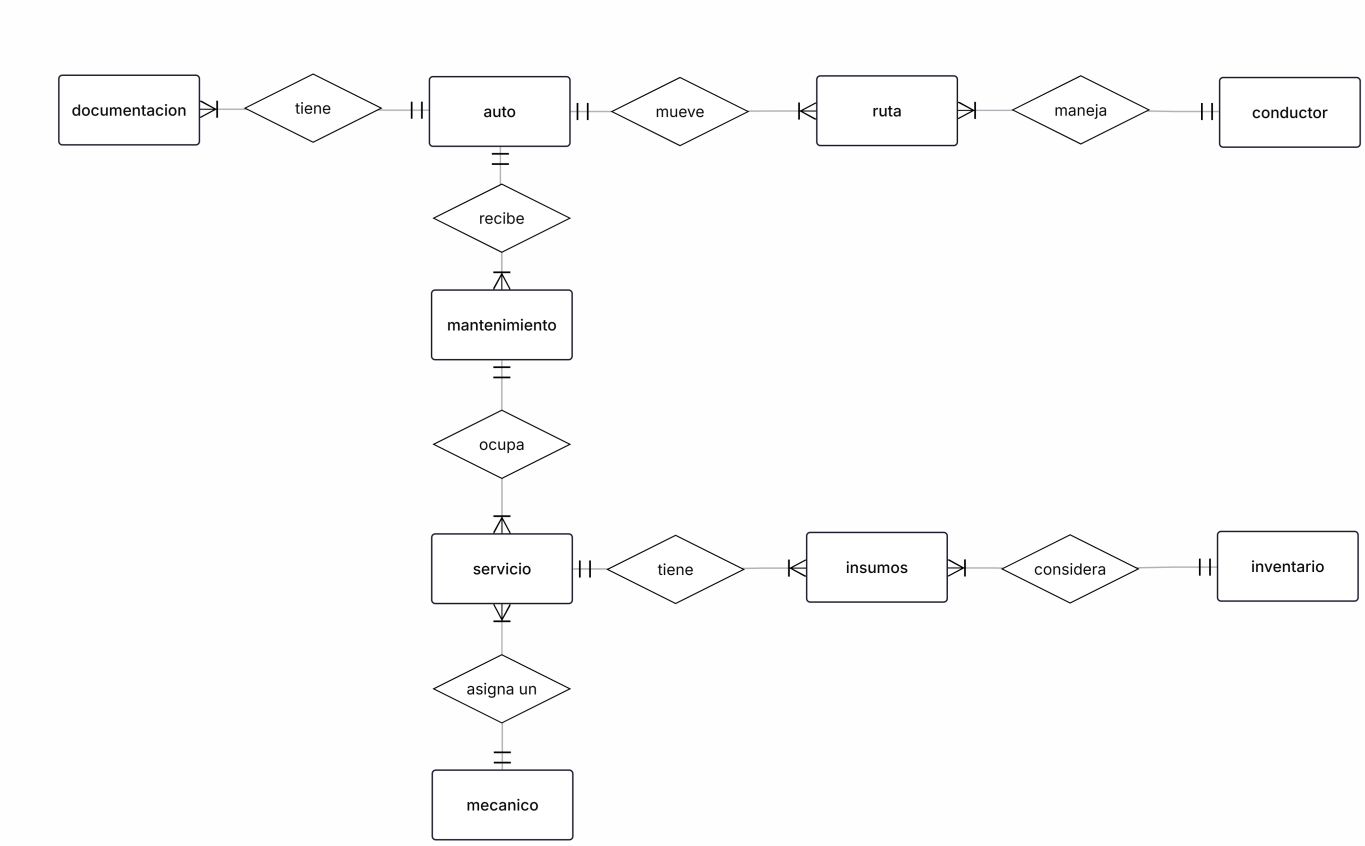


Figure 1: MER

# 3. Modelo Relacional

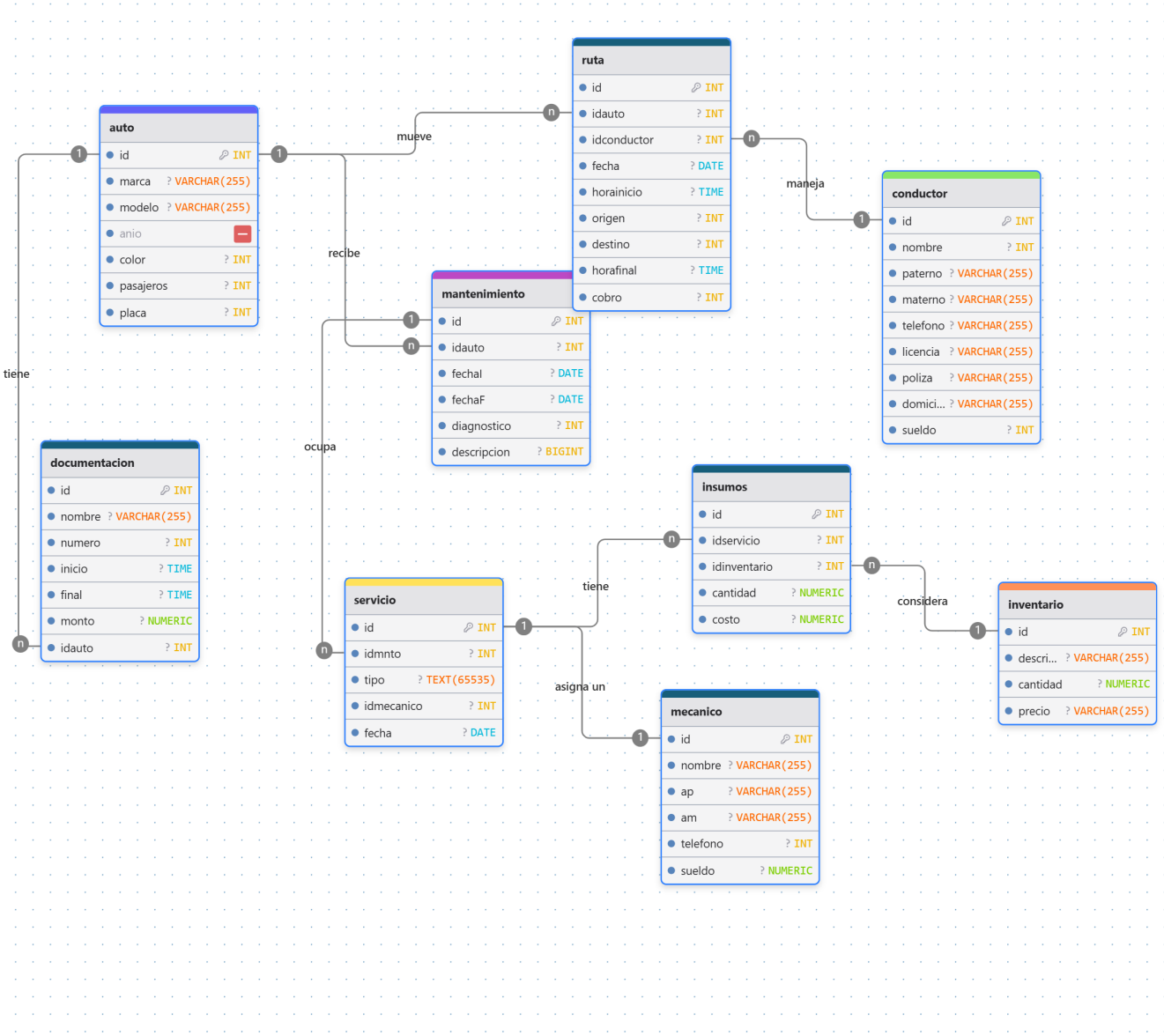


Figure 2: MR

# 4. Esquema conceptual local

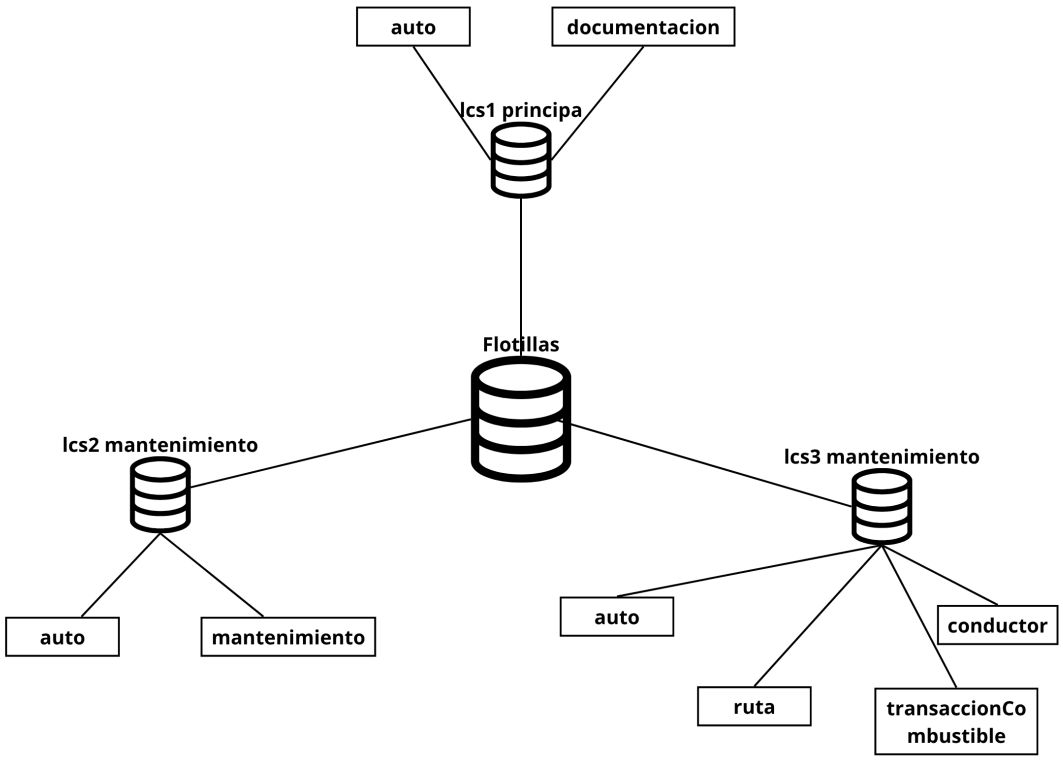


Figure 3: MR

# 5. Script de creación de nodos

## lcs1_principal

```
CREATE DATABASE lcs1_principal;
USE lcs1_principal;
CREATE TABLE auto (
id INT NOT NULL,
marca VARCHAR(255),
modelo VARCHAR(255),
anio INT,
color VARCHAR(255),
pasajeros INT,
placa VARCHAR(255),
PRIMARY KEY(id)
);
CREATE TABLE documentacion (
id INT NOT NULL,
nombre VARCHAR(255),
numero INT,
inicio DATE,
final DATE,
monto NUMERIC,
idAuto INT,
PRIMARY KEY(id),
FOREIGN KEY (idAuto) REFERENCES auto(id)
);
```

## lcs2_mantenimiento

```
CREATE DATABASE lcs2_mantenimiento;
USE lcs2_mantenimiento;
CREATE TABLE auto (
id INT NOT NULL,
marca VARCHAR(255),
modelo VARCHAR(255),
anio INT,
color VARCHAR(255),
pasajeros INT,
placa VARCHAR(255),
PRIMARY KEY(id)
);
CREATE TABLE mantenimiento (
id INT NOT NULL,
fechaInicio DATE,
fechaFinal DATE,
diagnostico VARCHAR(255),
descripcion VARCHAR(255),
idAuto INT,
PRIMARY KEY(id),
FOREIGN KEY (idAuto) REFERENCES auto(id)
);
```

## lcs3_rutas

```
CREATE DATABASE lcs3_rutas;
USE lcs3_rutas;
CREATE TABLE auto (
id INT NOT NULL,
marca VARCHAR(255),
modelo VARCHAR(255),
anio INT,
color VARCHAR(255),
pasajeros INT,
placa VARCHAR(255),
PRIMARY KEY(id)
); CREATE TABLE conductor (
id INT NOT NULL,
nombre VARCHAR(255),
apellidoPaterno VARCHAR(255),
apellidoMaterno VARCHAR(255),
telefono VARCHAR(255),
licencia VARCHAR(255),
poliza VARCHAR(255),
domicilio VARCHAR(255),
sueldo DECIMAL,
PRIMARY KEY(id)
);
CREATE TABLE ruta (
id INT NOT NULL,
fecha DATE,
horaInicio TIME,
origen VARCHAR(255),
destino VARCHAR(255),
horaFinal TIME,
cobro DECIMAL,
idConductor INT,
idAuto INT,
PRIMARY KEY(id),
FOREIGN KEY (idConductor) REFERENCES conductor(id),
FOREIGN KEY (idAuto) REFERENCES auto(id)
);
CREATE TABLE transaccionCombustible (
id INT NOT NULL,
idAuto INT,
fecha DATE,
cantidadLitros DECIMAL(10, 2),
costo DECIMAL(10, 2),
proveedor VARCHAR(255),
PRIMARY KEY(id),
FOREIGN KEY (idAuto) REFERENCES auto(id)
);
```

# 6. Scripts de extracción de datos

## lcs1_principal

```
SELECT * FROM auto;
SELECT * FROM documentacion;
```

## lcs2_mantenimiento

```
SELECT * FROM auto;
SELECT * FROM mantenimiento;
```

## lcs3_mantenimiento

```
SELECT * FROM auto;
SELECT * FROM conductor;
SELECT * FROM ruta;
SELECT * FROM transaccionCombustible;
```

# 7. Script de carga de datos

## lcs1_principal

load data local infile 'C:\ProgramData\MySQL\MySQL Server 8.0\Data\lcs1_principal\auto.csv'
ignore into table lcs1_principal.auto
fields terminated by ';'
enclosed by ' '
lines terminated by ''
ignore 1 lines
(id, marca, modelo, anio, color, pasajeros, placa);
load data local infile 'C:\ProgramData\MySQL\MySQL Server 8.0\Data\lcs1_principal\documentacion.csv'
ignore into table lcs1_principal.documentacion
fields terminated by ';'
enclosed by ' '
lines terminated by ''
ignore 1 lines
(id, nombre, numero, inicio, final, monto, idAuto);

## lcs3_mantenimiento

load data local infile 'C:\ProgramData\MySQL\MySQL Server 8.0\Data\lcs3_rutas\auto.csv'
ignore into table lcs3_rutas.auto
fields terminated by ';'
enclosed by ' '
lines terminated by ''
ignore 1 lines
(id, marca, modelo, anio, color, pasajeros, placa);
load data local infile 'C:\ProgramData\MySQL\MySQL Server 8.0\Data\lcs3_rutas\conductor.csv'
ignore into table lcs3_rutas.conductor
fields terminated by ';'
enclosed by ' '
lines terminated by ''
ignore 1 lines
(id, nombre, apellidoPaterno, apellidoMaterno, telefono, licencia, poliza, domicilio, sueldo);
load data local infile 'C:\ProgramData\MySQL\MySQL Server 8.0\Data\lcs3_rutas\ruta.csv'
ignore into table lcs3_rutas.ruta
fields terminated by ';'
enclosed by ' '
lines terminated by ''
ignore 1 lines
(id, fecha, horaInicio, origen, destino, horaFinal, cobro, idConductor, idAuto);
load data local infile 'C:\ProgramData\MySQL\MySQL Server 8.0\Data\lcs3_rutas\transaccionCombustible
.csv'
ignore into table lcs3_rutas.transaccionCombustible
fields terminated by ';'
enclosed by ' '
lines terminated by ''
ignore 1 lines
(id, idAuto, fecha, cantidadLitros, costo, proveedor);

# 8. Script de consulta de datos

## lcs1_principal

```
USE lcs1_principal;
SELECT
a.id,
a.marca,
a.modelo,
a.anio,
d.nombre AS tipo_documento,
d.numero,
d.inicio,
d.final
FROM auto a
LEFT JOIN documentacion d ON a.id = d.idAuto;
```

## lcs3_mantenimiento

```
USE lcs3_rutas;
SELECT
r.id,
r.fecha,
r.horaInicio,
r.origen,
r.destino,
r.horaFinal,
r.cobro,
c.nombre AS nombre_conductor,
c.apellidoPaterno,
a.marca,
a.modelo,
a.placa
FROM ruta r
INNER JOIN conductor c ON r.idConductor = c.id
INNER JOIN auto a ON r.idAuto = a.id;
```