Universidad Autónoma del Estado de Hidalgo

REPORTE DE PRÁCTICA NO. 1

Fragmentación en una Base de Datos

ALUMNO:

Jordan Gael Sosa De la Cruz
Hassiel Camacho Meneses

1. Introducción

La presente práctica se centra en el diseño, construcción y fragmentación de una base de datos relacional para la gestión de una flotilla de autos. El objetivo es estructurar un modelo que permita almacenar y organizar información sobre vehículos, conductores, propietarios, mecánicos, mantenimientos, insumos, inventario, rutas y documentos.
Una vez diseñado el modelo, se aplicó el concepto de fragmentación de base de datos para dividir las entidades en segmentos asignados a distintos tipos de usuarios. Esta técnica permite distribuir la información de manera lógica y controlada, mejorando la seguridad, el acceso a datos específicos y la administración general del sistema.

2. Marco Teórico

Las bases de datos son sistemas diseñados para almacenar y gestionar grandes volúmenes de información de manera estructurada y eficiente. Tradicionalmente, los modelos centralizados concentraban toda la información en un único servidor; sin embargo, este enfoque presenta limitaciones en cuanto a disponibilidad, escalabilidad y rendimiento. Como respuesta, surgieron las **bases de datos distribuidas**, en las que la información se divide en distintos fragmentos que pueden almacenarse en diferentes nodos, pero que en conjunto se perciben como una sola base de datos unificada.
Un sistema de bases de datos distribuidas tiene como objetivo mantener propiedades esenciales como la **consistencia de la información**, la **integridad de los datos** y la **disponibilidad del sistema**, incluso cuando los fragmentos están dispersos en diversas ubicaciones físicas. Para ello, se emplean técnicas como la replicación, el control de concurrencia y la sincronización de transacciones.
La **fragmentación** constituye un aspecto clave en este tipo de sistemas. Se refiere al proceso de dividir la base de datos en subconjuntos más pequeños llamados *fragmentos*, lo que permite asignar a cada usuario o nodo únicamente la parte de la información que necesita. Existen tres tipos de fragmentación:

- **Horizontal**: divide las tablas en subconjuntos de registros basados en condiciones lógicas, por ejemplo, asignar automóviles de acuerdo con la zona en la que operan.
- **Vertical**: separa las tablas por atributos o columnas, conservando siempre la llave primaria para garantizar que los fragmentos puedan reconstruirse.
- **Híbrida**: combina la fragmentación horizontal y vertical para responder a necesidades más complejas de seguridad y desempeño.

Entre los beneficios de la fragmentación se encuentran el aumento en la **eficiencia de las consultas**, ya que los usuarios acceden únicamente a los datos que requieren; el refuerzo de la **seguridad**, pues se limita la exposición de información sensible; la **escalabilidad**, al permitir añadir nuevos fragmentos sin alterar la estructura global; y la **disponibilidad**, dado que cada nodo puede seguir funcionando incluso si otros presentan fallas.
En el diseño de bases de datos distribuidas también es posible simplificar cada fragmento eliminando claves foráneas innecesarias, con el fin de reducir dependencias y hacer que cada segmento funcione de manera más autónoma. Aun así, la integridad del sistema completo se preserva mediante la correcta definición de las relaciones entre fragmentos.
Un escenario potencial es que los fragmentos se almacenen en la nube y sean actualizados a través de aplicaciones móviles. En caso de desconexión de alguno de los nodos, la información puede seguir siendo gestionada de forma local y, al restablecerse la conexión, sincronizarse con la base de datos central almacenada en la nube. De este modo, cada fragmento mantiene su autonomía operativa, pero contribuye a la coherencia de un sistema global.

Procedimientos almacenados

Los procedimientos almacenados son bloques de código guardados en el servidor de la base de datos que permiten ejecutar una serie de instrucciones de forma encapsulada. Facilitan la automatización de tareas y garantizan que la lógica de negocio se ejecute siempre de la misma manera.

Listing 1: Ejemplo de procedimiento almacenado

```
CREATE PROCEDURE registrarAuto(
  IN pMarca VARCHAR(50), IN pModelo VARCHAR(50))
BEGIN
  INSERT INTO auto(marca, modelo) VALUES(pMarca, pModelo);
END;
```

Funciones

Las funciones devuelven un valor único y pueden ser usadas dentro de consultas para cálculos frecuentes. Son útiles para operaciones que se repiten, como obtener fechas, sumas o estados.

Listing 2: Ejemplo de función

```
CREATE FUNCTION totalServicios(p_idAuto INT)
RETURNS INT
BEGIN
  RETURN (SELECT COUNT(*) FROM servicio WHERE idAuto = p_idAuto);
END;
```

Estructuras de control

Las estructuras de control permiten tomar decisiones y repetir acciones dentro de rutinas SQL. Incluyen condicionales (`IF`) y bucles (`WHILE`).

Listing 3: Ejemplo de estructura IF

```
IF (SELECT COUNT(*) FROM documento
    WHERE idAuto = 1 AND final < CURDATE()) > 0
THEN
  INSERT INTO alertas VALUES('Documento vencido');
END IF;
```

Listing 4: Ejemplo de estructura WHILE

```
SET @i = 1;
WHILE @i <= 5 DO
  INSERT INTO revisiones(auto) VALUES(@i);
  SET @i = @i + 1;
END WHILE;
```

Disparadores (Triggers)

Los disparadores son rutinas que se ejecutan automáticamente al producirse un evento como `INSERT`, `UPDATE` o `DELETE`. Se usan para auditoría o para mantener integridad.

Listing 5: Ejemplo de trigger

```
CREATE TRIGGER logRuta
AFTER UPDATE ON ruta
FOR EACH ROW
INSERT INTO auditoria
VALUES(NOW(), OLD.idConductor, NEW.idConductor);
```

3. Herramientas empleadas

DrawDatabase: Utilizada para diseñar y visualizar el modelo entidad-relación y relacional. Permitió organizar las entidades y sus relaciones, además de servir como base para aplicar la fragmentación.

ERDPlus: Herramienta inicial para generar los modelos y obtener código SQL a partir del diseño.

MySQL Server: Usado para implementar las tablas, llaves primarias y foráneas, además de validar el funcionamiento del modelo y simular la fragmentación a través de vistas o esquemas específicos para cada usuario.

4. Desarrollo

1. **Análisis de requerimientos:** Se definieron las entidades necesarias (auto, conductor, propietario, ruta, documentos, mantenimiento, servicio, mecánico, insumo e inventario).
2. **Diseño del modelo entidad–relación y modelo relacional:** Se representaron gráficamente las entidades y sus relaciones.
3. **Implementación en MySQL:** Se crearon las tablas, atributos y relaciones. Se verificó su integridad mediante consultas de prueba.
4. **Fragmentación de la base de datos:** Se segmentó la información en función de los usuarios del sistema:

- **Administrador:** acceso a todas las tablas del sistema (control total).

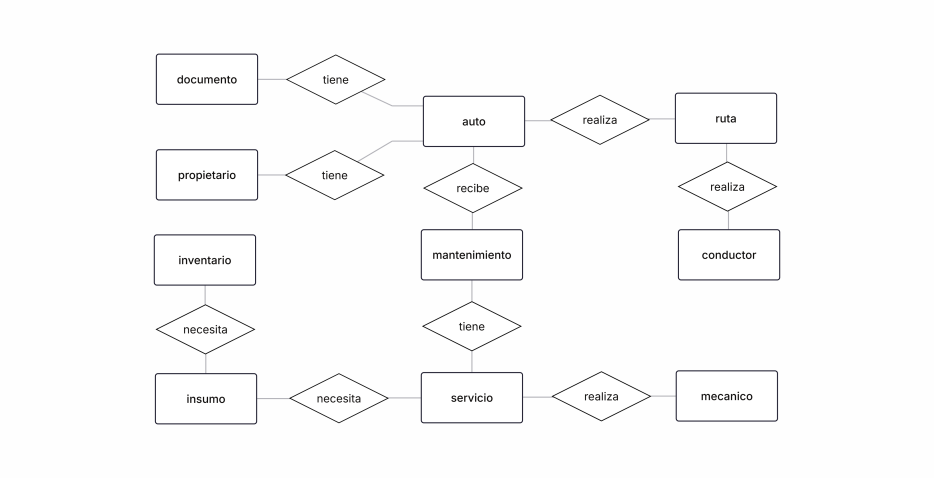

Figure 1: Fragmento Administrador ER

Figure 2: Fragmento Administrador MER

Listing 6: Fragmento de la base de datos de gestión de flotillas

```
CREATE DATABASE fragmento_propietario;
USE fragmento_propietario;

CREATE TABLE propietario (
    idPropietario INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    nombre VARCHAR(255) NOT NULL,
    paterno VARCHAR(255) NOT NULL,
```

```
    materno VARCHAR(255),
    PRIMARY KEY(idPropietario)
);

CREATE TABLE auto (
    idAuto INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idPropietario INTEGER NOT NULL,
    marca VARCHAR(255) NOT NULL,
    modelo VARCHAR(255) NOT NULL,
    anio INTEGER NOT NULL,
    color VARCHAR(255) NOT NULL,
    pasajeros INTEGER NOT NULL,
    placa VARCHAR(255) NOT NULL,
    PRIMARY KEY(idAuto),
    FOREIGN KEY(idPropietario) REFERENCES propietario(idPropietario)
);

CREATE TABLE ruta (
    idRuta INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idAuto INTEGER NOT NULL,
    idConductor INTEGER NOT NULL,
    fecha DATE NOT NULL,
    horaInicio TIME NOT NULL,
    horaFinal TIME NOT NULL,
    cobro DOUBLE NOT NULL,
    PRIMARY KEY(idRuta),
    FOREIGN KEY(idAuto) REFERENCES auto(idAuto),
    FOREIGN KEY(idConductor) REFERENCES conductor(idConductor)
);

CREATE TABLE documento (
    idDocumento INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idAuto INTEGER NOT NULL,
    nombre VARCHAR(255) NOT NULL,
    numero VARCHAR(255),
    inicio DATE,
    final DATE,
    monto DOUBLE NOT NULL,
    PRIMARY KEY(idDocumento),
    FOREIGN KEY(idAuto) REFERENCES auto(idAuto)
);

CREATE TABLE conductor (
    idConductor INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    nombre VARCHAR(255) NOT NULL,
    paterno VARCHAR(255) NOT NULL,
    materno VARCHAR(255),
    telefono VARCHAR(255) NOT NULL,
    licencia VARCHAR(255) NOT NULL,
    poliza VARCHAR(255) NOT NULL,
    domicilio VARCHAR(255) NOT NULL,
    sueldo DOUBLE NOT NULL,
    PRIMARY KEY(idConductor)
);
```

```
CREATE TABLE mantenimiento (
   idMantenimiento INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
   idAuto INTEGER NOT NULL,
   fechaInicio DATE NOT NULL,
   fechaFinal DATE NOT NULL,
   diagnostico VARCHAR(255),
   descripcion VARCHAR(255),
  PRIMARY KEY(idMantenimiento),
  FOREIGN KEY(idAuto) REFERENCES auto(idAuto)
);

CREATE TABLE inventario (
   idInventario INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
   descripcion VARCHAR(255) NOT NULL,
   cantidad INTEGER NOT NULL,
   precio DOUBLE,
  PRIMARY KEY(idInventario)
);

CREATE TABLE mecanico (
   idMecanico INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
   nombre VARCHAR(255) NOT NULL,
   paterno VARCHAR(255) NOT NULL,
   materno VARCHAR(255),
   telefono VARCHAR(255) NOT NULL,
   sueldo DOUBLE NOT NULL,
  PRIMARY KEY(idMecanico)
);

CREATE TABLE servicio (
   idServicio INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
   idMantenimiento INTEGER NOT NULL,
   idMecanico INTEGER NOT NULL,
   tipo VARCHAR(255) NOT NULL,
   fecha DATE NOT NULL,
  PRIMARY KEY(idServicio),
  FOREIGN KEY(idMantenimiento) REFERENCES mantenimiento(idMantenimiento),
  FOREIGN KEY(idMecanico) REFERENCES mecanico(idMecanico)
);

CREATE TABLE insumo (
   idInsumo INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
   idServicio INTEGER NOT NULL,
   idInventario INTEGER NOT NULL,
   cantidad INTEGER NOT NULL,
   costo DOUBLE NOT NULL,
  PRIMARY KEY(idInsumo),
  FOREIGN KEY(idInventario) REFERENCES inventario(idInventario),
  FOREIGN KEY(idServicio) REFERENCES servicio(idServicio)
);
```

- **Conductor:** acceso únicamente a las tablas *conductor, ruta y auto.*

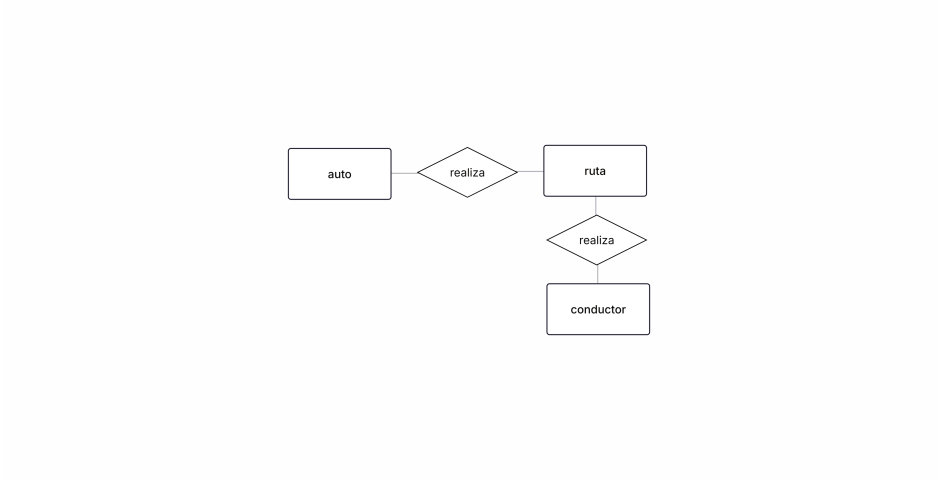

Figure 3: Fragmento Conductor ER

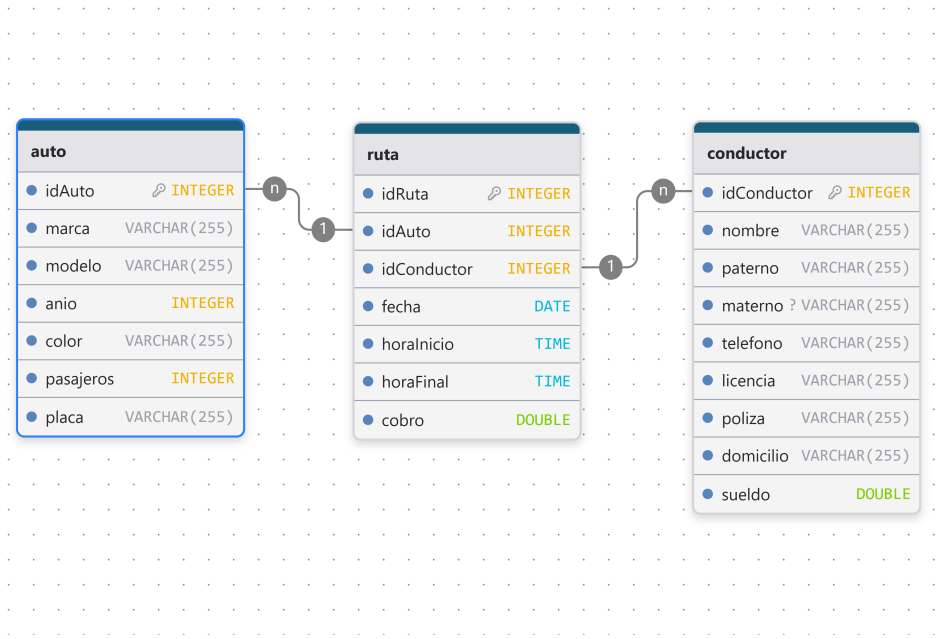

Figure 4: Fragmento Conductor MER

Listing 7: Fragmento de la base de datos: Conductor

```
CREATE DATABASE fragmento_conductor;
USE fragmento_conductor;

CREATE TABLE auto (
    idAuto INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    marca VARCHAR(255) NOT NULL,
    modelo VARCHAR(255) NOT NULL,
    anio INTEGER NOT NULL,
```

```
  color VARCHAR(255) NOT NULL,
  pasajeros INTEGER NOT NULL,
  placa VARCHAR(255) NOT NULL,
  PRIMARY KEY(idAuto)
);

CREATE TABLE ruta (
  idRuta INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
  idAuto INTEGER NOT NULL,
  idConductor INTEGER NOT NULL,
  fecha DATE NOT NULL,
  horaInicio TIME NOT NULL,
  horaFinal TIME NOT NULL,
  cobro DOUBLE NOT NULL,
  PRIMARY KEY(idRuta),
  FOREIGN KEY(idAuto) REFERENCES auto(idAuto),
  FOREIGN KEY(idConductor) REFERENCES conductor(idConductor)
);

CREATE TABLE conductor (
  idConductor INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
  nombre VARCHAR(255) NOT NULL,
  paterno VARCHAR(255) NOT NULL,
  materno VARCHAR(255),
  telefono VARCHAR(255) NOT NULL,
  licencia VARCHAR(255) NOT NULL,
  poliza VARCHAR(255) NOT NULL,
  domicilio VARCHAR(255) NOT NULL,
  sueldo DOUBLE NOT NULL,
  PRIMARY KEY(idConductor)
);
```

- **Mecánico:** acceso a *inventario, insumo, mantenimiento, servicio, mecánico y auto.*

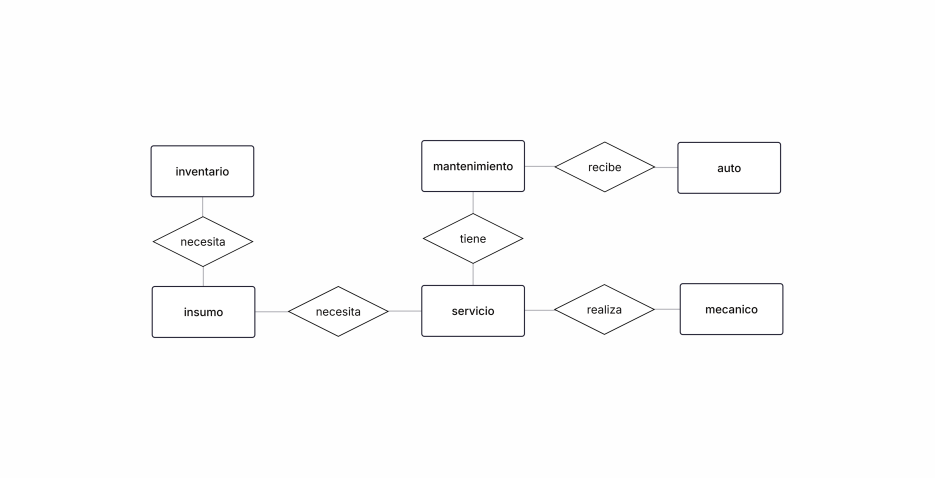

Figure 5: Fragmento Mecánico ER

Figure 6: Fragmento Mecánico MER

```
CREATE DATABASE fragmento_mecanico;
USE fragmento_mecanico;

CREATE TABLE auto (
    idAuto INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    marca VARCHAR(255) NOT NULL,
    modelo VARCHAR(255) NOT NULL,
    anio INTEGER NOT NULL,
    color VARCHAR(255) NOT NULL,
    pasajeros INTEGER NOT NULL,
```

```
    placa VARCHAR(255) NOT NULL,
    PRIMARY KEY(idAuto),
);

CREATE TABLE mantenimiento (
    idMantenimiento INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idAuto INTEGER NOT NULL,
    fechaInicio DATE NOT NULL,
    fechaFinal DATE NOT NULL,
    diagnostico VARCHAR(255),
    descripcion VARCHAR(255),
    PRIMARY KEY(idMantenimiento),
    FOREIGN KEY(idAuto) REFERENCES auto(idAuto)
);

CREATE TABLE inventario (
    idInventario INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    descripcion VARCHAR(255) NOT NULL,
    cantidad INTEGER NOT NULL,
    precio DOUBLE,
    PRIMARY KEY(idInventario)
);

CREATE TABLE mecanico (
    idMecanico INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    nombre VARCHAR(255) NOT NULL,
    paterno VARCHAR(255) NOT NULL,
    materno VARCHAR(255),
    telefono VARCHAR(255) NOT NULL,
    sueldo DOUBLE NOT NULL,
    PRIMARY KEY(idMecanico)
);

CREATE TABLE servicio (
    idServicio INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idMantenimiento INTEGER NOT NULL,
    idMecanico INTEGER NOT NULL,
    tipo VARCHAR(255) NOT NULL,
    fecha DATE NOT NULL,
    PRIMARY KEY(idServicio),
    FOREIGN KEY(idMantenimiento) REFERENCES mantenimiento(idMantenimiento),
    FOREIGN KEY(idMecanico) REFERENCES mecanico(idMecanico)
);

CREATE TABLE insumo (
    idInsumo INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idServicio INTEGER NOT NULL,
    idInventario INTEGER NOT NULL,
    cantidad INTEGER NOT NULL,
    costo DOUBLE NOT NULL,
    PRIMARY KEY(idInsumo),
    FOREIGN KEY(idInventario) REFERENCES inventario(idInventario),
    FOREIGN KEY(idServicio) REFERENCES servicio(idServicio)
);
```

- **Propietario:** acceso a *auto, documentos, rutas y propietario.*

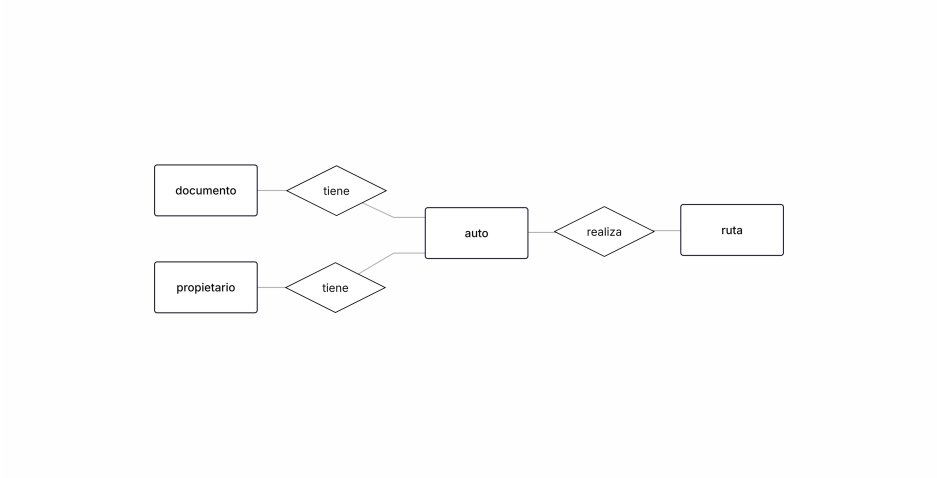

Figure 7: Fragmento Propietario ER

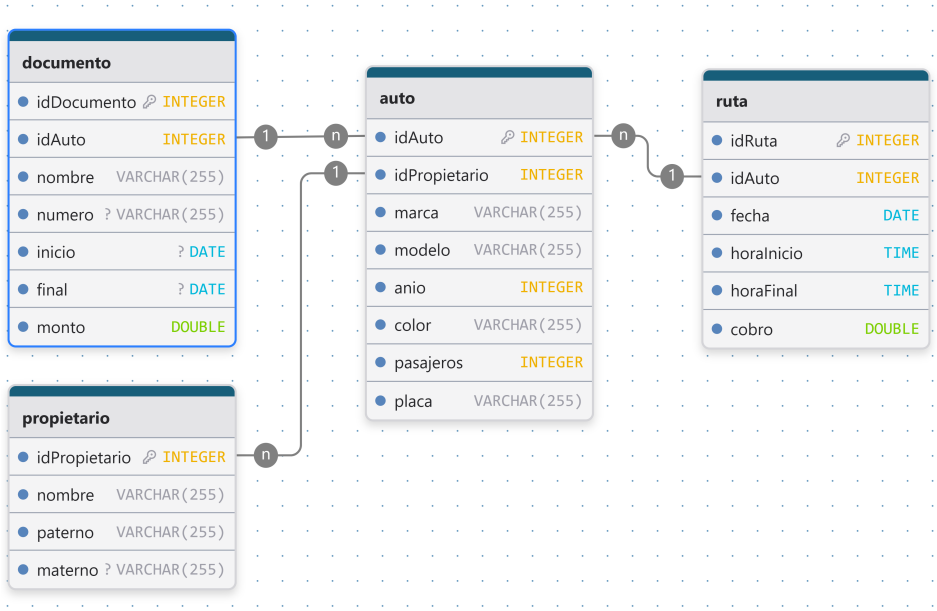

Figure 8: Fragmento Propietario MER

```
CREATE DATABASE fragmento_propietario;
USE fragmento_propietario;

CREATE TABLE propietario (
    idPropietario INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    nombre VARCHAR(255) NOT NULL,
    paterno VARCHAR(255) NOT NULL,
    materno VARCHAR(255),
    PRIMARY KEY(idPropietario)
);
```

```
CREATE TABLE auto (
    idAuto INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idPropietario INTEGER NOT NULL,
    marca VARCHAR(255) NOT NULL,
    modelo VARCHAR(255) NOT NULL,
    anio INTEGER NOT NULL,
    color VARCHAR(255) NOT NULL,
    pasajeros INTEGER NOT NULL,
    placa VARCHAR(255) NOT NULL,
    PRIMARY KEY(idAuto),
    FOREIGN KEY(idPropietario) REFERENCES propietario(idPropietario)
);

CREATE TABLE ruta (
    idRuta INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idAuto INTEGER NOT NULL,
    fecha DATE NOT NULL,
    horaInicio TIME NOT NULL,
    horaFinal TIME NOT NULL,
    cobro DOUBLE NOT NULL,
    PRIMARY KEY(idRuta),
    FOREIGN KEY(idAuto) REFERENCES auto(idAuto)
);

CREATE TABLE documento (
    idDocumento INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idAuto INTEGER NOT NULL,
    nombre VARCHAR(255) NOT NULL,
    numero VARCHAR(255),
    inicio DATE,
    final DATE,
    monto DOUBLE NOT NULL,
    PRIMARY KEY(idDocumento),
    FOREIGN KEY(idAuto) REFERENCES auto(idAuto)
);
```

Esta fragmentación se implementó mediante la creación de diferentes vistas y permisos de usuario en MySQL, de forma que cada rol solo consulta o manipula los datos que le corresponden.

Modelo Entidad - Relación

Descripción de los atributos:
propietario (idPropietario, nombre, paterno, materno) **auto** (idAuto, idPropietario, marca, modelo, anio, color, pasajeros, placa) **ruta** (idRuta, idAuto, idConductor, fecha, horaInicio, horaFinal, cobro) **documento** (idDocumento, idAuto, nombre, numero, inicio, final, monto) **conductor** (idConductor, nombre, paterno, materno, telefono, licencia, poliza, domicilio, sueldo) **mantenimiento** (idMantenimiento, idAuto, fechaInicio, fechaFinal, diagnostico, descripcion) **inventario** (idInventario, descripcion, cantidad, precio) **mecanico** (idMecanico, nombre, paterno, materno, telefono, sueldo) **servicio** (idServicio, idMantenimiento, idMecanico, tipo, fecha) **insumo** (idInsumo, idServicio, idInventario, cantidad, costo)

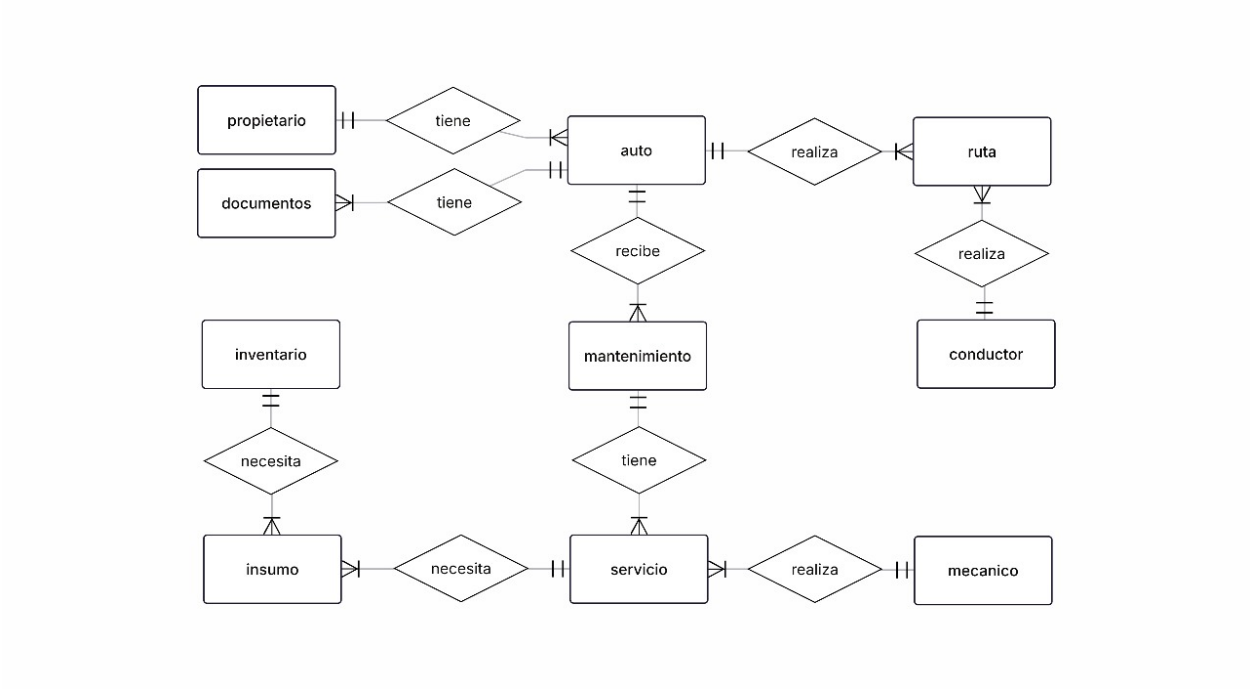

Figure 9: Modelo MER

Modelo relacional

Figure 10: Modelo MR

SQL

Listing 8: Descripción de comandos SQL

```
CREATE DATABASE flotilla;

USE flotilla;

CREATE TABLE propietario (
        idPropietario INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
        nombre VARCHAR(255) NOT NULL,
        paterno VARCHAR(255) NOT NULL,
        materno VARCHAR(255),
       PRIMARY KEY(idPropietario)
);

CREATE TABLE auto (
        idAuto INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
        idPropietario INTEGER NOT NULL,
```

```
        marca VARCHAR(255) NOT NULL,
        modelo VARCHAR(255) NOT NULL,
        anio INTEGER NOT NULL,
        color VARCHAR(255) NOT NULL,
        pasajeros INTEGER NOT NULL,
        placa VARCHAR(255) NOT NULL,
        PRIMARY KEY(idAuto),
        FOREIGN KEY(idPropietario) REFERENCES propietario(idPropietario)
);

CREATE TABLE ruta (
        idRuta INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
        idAuto INTEGER NOT NULL,
        idConductor INTEGER NOT NULL,
        fecha DATE NOT NULL,
        horaInicio TIME NOT NULL,
        horaFinal TIME NOT NULL,
        cobro DOUBLE NOT NULL,
        PRIMARY KEY(idRuta),
        FOREIGN KEY(idAuto) REFERENCES auto(idAuto),
        FOREIGN KEY(idConductor) REFERENCES conductor(idConductor)
);

CREATE TABLE documento (
        idDocumento INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
        idAuto INTEGER NOT NULL,
        nombre VARCHAR(255) NOT NULL,
        numero VARCHAR(255),
        inicio DATE,
        final DATE,
        monto DOUBLE NOT NULL,
        PRIMARY KEY(idDocumento),
        FOREIGN KEY(idAuto) REFERENCES auto(idAuto)
);

CREATE TABLE conductor (
        idConductor INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
        nombre VARCHAR(255) NOT NULL,
        paterno VARCHAR(255) NOT NULL,
        materno VARCHAR(255),
        telefono VARCHAR(255) NOT NULL,
        licencia VARCHAR(255) NOT NULL,
        poliza VARCHAR(255) NOT NULL,
        domicilio VARCHAR(255) NOT NULL,
        sueldo DOUBLE NOT NULL,
        PRIMARY KEY(idConductor)
);

CREATE TABLE mantenimiento (
        idMantenimiento INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
        idAuto INTEGER NOT NULL,
        fechaInicio DATE NOT NULL,
        fechaFinal DATE NOT NULL,
        diagnostico VARCHAR(255),
```

```
    descripcion VARCHAR(255),
    PRIMARY KEY(idMantenimiento),
    FOREIGN KEY(idAuto) REFERENCES auto(idAuto)
);

CREATE TABLE inventario (
    idInventario INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    descripcion VARCHAR(255) NOT NULL,
    cantidad INTEGER NOT NULL,
    precio DOUBLE,
    PRIMARY KEY(idInventario)
);

CREATE TABLE mecanico (
    idMecanico INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    nombre VARCHAR(255) NOT NULL,
    paterno VARCHAR(255) NOT NULL,
    materno VARCHAR(255),
    telefono VARCHAR(255) NOT NULL,
    sueldo DOUBLE NOT NULL,
    PRIMARY KEY(idMecanico)
);

CREATE TABLE servicio (
    idServicio INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idMantenimiento INTEGER NOT NULL,
    idMecanico INTEGER NOT NULL,
    tipo VARCHAR(255) NOT NULL,
    fecha DATE NOT NULL,
    PRIMARY KEY(idServicio),
    FOREIGN KEY(idMantenimiento) REFERENCES mantenimiento(idMantenimiento),
    FOREIGN KEY(idMecanico) REFERENCES mecanico(idMecanico)
);


CREATE TABLE insumo (
    idInsumo INTEGER NOT NULL AUTO_INCREMENT UNIQUE,
    idServicio INTEGER NOT NULL,
    idInventario INTEGER NOT NULL,
    cantidad INTEGER NOT NULL,
    costo DOUBLE NOT NULL,
    PRIMARY KEY(idInsumo),
    FOREIGN KEY(idInventario) REFERENCES inventario(idInventario),
    FOREIGN KEY(idServicio) REFERENCES servicio(idServicio)
);
```

Proceso ETL para poblar la Base de Datos Global

1. Extracción (Extract)

Cada fragmento contiene únicamente la información correspondiente a su rol:

- **Fragmento Conductor:** tablas `conductor`, `ruta`, `auto`.

- **Fragmento Mecánico:** tablas `mecanico`, `mantenimiento`, `inventario`, `insumo`, `servicio`, `auto`.
- **Fragmento Propietario:** tablas `propietario`, `auto`, `documento`, `ruta`.

En esta fase, se extraen los datos de cada fragmento mediante consultas SQL `SELECT` o vistas que centralizan la información.

```
SELECT * FROM conductor;
SELECT * FROM ruta;
```

2. Transformación (Transform)

Como los fragmentos eliminan llaves foráneas innecesarias para ser más autónomos, antes de integrarlos al fragmento del administrador se deben normalizar y unificar:

- **Conversión de formatos:** asegurar que los atributos (`anio`, `fecha`, `monto`, etc.) usen el mismo tipo de dato en todos los fragmentos.
- **Reconstrucción de relaciones:** volver a conectar las tablas mediante sus claves primarias y foráneas, de acuerdo al modelo global (MER y MR).
- **Eliminación de redundancias:** por ejemplo, si el `auto` aparece en varios fragmentos, se consolida en una sola tabla de la base de datos global.
- **Homologación de nombres:** atributos como `anio/an io`, `costo/cos to` deben estandarizarse.

Ejemplo de transformación:

```
INSERT INTO flotilla.auto
(idAuto, idPropietario, marca, modelo, anio, color, pasajeros, placa)
SELECT idAuto, idPropietario, marca, modelo, anio, color, pasajeros, placa
FROM fragmento_propietario.auto;
```

3. Carga (Load)

Finalmente, los datos transformados se cargan en el **fragmento administrador**, que actúa como la base de datos global.
El administrador tiene todas las tablas: `auto`, `conductor`, `propietario`, `documento`, `mantenimiento`, `inventario`, `mecanico`, `servicio`, `insumo`, `ruta`.
Se emplean sentencias INSERT INTO SELECT o procesos más automatizados mediante **procedimientos almacenados** para sincronizar cada fragmento con la base central.
Ejemplo:

```
INSERT INTO flotilla.conductor
SELECT * FROM fragmento_conductor.conductor;
```

5. MARCO TEÓRICO

1. Fragmentación Vertical

La **fragmentación vertical** consiste en dividir una tabla en subconjuntos de columnas. Cada fragmento conserva siempre la **llave primaria**, lo que permite reconstruir la tabla original mediante un **JOIN**.
Se utiliza cuando distintos usuarios o nodos necesitan solo ciertos atributos y no toda la tabla.
Ventajas:

- Reduce transferencia de datos
- Aumenta seguridad
- Mejora tiempos de consulta

Ejemplo:
`-- Fragmento vertical de tabla AUTO (solo datos del conductor) CREATE TABLE auto_conductor ( idAuto INT`

2. Procesos ETL (Extract, Transform, Load)

5.2.1 Extract (Extracción)

Obtiene los datos desde cada nodo fragmentado.
Ejemplo:
`SELECT * FROM fragmento_conductor.auto;`

5.2.2 Transform (Transformación)

Normaliza, limpia y homologa datos:

- Igualar tipos de datos
- Corregir nombres
- Reconstruir claves foráneas
- Quitar duplicados

5.2.3 Load (Carga)

Insertar en la **base global (administrador)**:
`INSERT INTO flotilla.auto SELECT * FROM fragmento_propietario.auto;`

3. SELECT ... INTO FILE

Sirve para **exportar datos a un archivo** desde una tabla o vista.
Ejemplo:
`SELECT * FROM auto INTO OUTFILE '/ruta/auto.csv' FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';` Usado normalmente en la fase "Extract" del ETL.

4. LOAD DATA INFILE

Permite **cargar datos desde un archivo externo hacia MySQL**, útil en la fase "Load".
Ejemplo:
`LOAD DATA INFILE '/ruta/auto.csv' INTO TABLE auto FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';`

5. SELECT entre tablas de dos bases diferentes

MySQL permite consultar dos o más bases si están en el mismo servidor:

```
SELECT a.marca, c.nombre FROM fragmento_propietario.auto a JOIN fragmento_conductor.conductor c ON a.id
```

ESQUEMA CONCEPTUAL LOCAL DE CADA NODO

Cada nodo tendrá acceso solo a las tablas necesarias según tu PDF.

6.0.1 1. Nodo Administrador

Acceso total a:

- propietario
- auto
- conductor
- ruta
- documento
- mecanico
- mantenimiento
- inventario
- servicio
- insumo

6.0.2 2. Nodo Conductor

- auto
- ruta
- conductor

6.0.3 3. Nodo Mecánico

- auto
- mantenimiento
- mecanico
- inventario
- servicio
- insumo

6.0.4 4. Nodo Propietario

- propietario
- auto
- documento
- ruta

SCRIPT DE CREACIÓN DE NODOS

Usando lo que traía tu PDF.

1. Script Nodo Administrador

```
CREATE DATABASE nodo_admin; USE nodo_admin; -- (Aquí pegas el SQL completo del administrador tal como ap
```

2. Script Nodo Conductor

```
CREATE DATABASE nodo_conductor; USE nodo_conductor; CREATE TABLE auto ( idAuto INT AUTO_INCREMENT PRIMAR
```

3. Script Nodo Mecánico

```
CREATE DATABASE nodo_mecanico; USE nodo_mecanico; CREATE TABLE auto (...); CREATE TABLE mantenimiento (
```

vienen completos en tu PDF)

4. Script Nodo Propietario

```
CREATE DATABASE nodo_propietario; USE nodo_propietario; CREATE TABLE propietario (...); CREATE TABLE aut
```

SCRIPTS ETL

1. Extracción

```
SELECT * FROM nodo_conductor.conductor; SELECT * FROM nodo_mecanico.mecanico; SELECT * FROM nodo_propiet
```

2. Transformación

Normalización de nombres y datos:

```
INSERT INTO nodo_admin.auto (idAuto, idPropietario, marca, modelo, anio, color, pasajeros, placa) SELECT
```

3. Carga

```
INSERT INTO nodo_admin.conductor SELECT * FROM nodo_conductor.conductor;
```

SCRIPT DE CONSULTA ENTRE DOS NODOS

Ejemplo solicitado:

```
SELECT nodo_conductor.conductor.nombre, nodo_propietario.auto.marca, nodo_propietario.auto.modelo FROM n
```

INSTRUCCIONES FINALES PARA ENTREGAR EN OVERLIFE

1. **Sube a tu repo**: Carpeta `/nodos/` con:
    - nodo_admin.sql
    - nodo_conductor.sql
    - nodo_mecanico.sql
    - nodo_propietario.sql
    - ETL_extract.sql
    - ETL_transform.sql

- ETL_load.sql
- consulta_multinodo.sql

2. **Copia la URL del repositorio**
3. **Pégala en Plataforma Garza**

6. Conclusiones

La práctica permitió consolidar conocimientos sobre diseño de bases de datos relacionales y, adicionalmente, comprender cómo aplicar la fragmentación para distribuir la información de manera lógica y segura.
La división de la base en segmentos por usuario facilitó identificar qué información requiere cada rol, evitando accesos innecesarios. Esto no solo mejora la seguridad de los datos, sino también la eficiencia en las consultas, ya que cada usuario trabaja únicamente con lo que necesita.
En general, el ejercicio reforzó la importancia de diseñar bases de datos completas y escalables, y a la vez mostró cómo la fragmentación es una técnica clave en sistemas distribuidos y de acceso compartido.

Referencias Bibliográficas

References

[1] Edenred México. (2023, 2 de octubre). *Flotilla de autos: cómo administrarla.* Edenred México. https://www.edenred.mx/blog/flotilla-de-autos-como-administrarla

[2] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Fundamentos de bases de datos* (5a ed.). McGraw-Hill.