REPORTE DE PRÁCTICA 2.4

NOMBRE DE LA PRÁCTICA: Práctica. 3 nodos BDD Flotillas

ALUMNO: Axel Damian Ortiz Simon
Dr. Eduardo Cornejo-Velázquez

1. Marco teórico

En esta práctica, se abordaron conceptos clave relacionados con la gestión de bases de datos distribuidas, incluyendo la fragmentación vertical, los procesos ETL, y el uso de comandos como 'SELECT ... INTO FILE', 'LOAD DATA INFILE', y consultas que involucran múltiples nodos y se presentará el esquema conceptual local de cada nodo junto con los scripts necesarios para la creación, extracción, carga y consulta de datos en un entorno distribuido.

1. Fragmentación vertical

La fragmentación vertical es un tipo de partición de datos en bases distribuidas que consiste en dividir una tabla en varias sub-tablas, conservando la clave primaria en cada una de ellas. Su objetivo es mejorar el rendimiento del sistema al distribuir los datos de manera eficiente, permitiendo consultas más rápidas y optimizadas en cada fragmento.

Ejemplo de fragmentación vertical

Si tenemos una tabla `Vehiculo(id, modelo, año, propietario, placa)`, podríamos dividirla en dos fragmentos:

- `Vehiculo_Identificacion(id, placa)`
- `Vehiculo_Detalle(id, modelo, año, propietario)`

Esto permite consultas optimizadas dependiendo del tipo de información requerida.

2. Procesos ETL (Extract, Transform, Load)

Los procesos ETL son utilizados para extraer datos de una fuente, transformarlos según sea necesario y cargarlos en un destino. En esta práctica, se utilizaron procesos ETL para:

- **Extraer:** Obtención de datos desde diversas fuentes (GCS) usando 'SELECT ... INTO FILE'.
- **Transformar:** Limpieza y adaptación de los datos para cumplir con los requisitos del sistema destino. Ajustar el formato de los datos para su carga en las bases de datos locales.
- **Cargar:** Inserción de los datos en la base de datos final o en los nodos correspondientes locales usando 'LOAD DATA INFILE'.

3. SELECT + INTO FILE

El comando 'SELECT ... INTO OUTFILE' permite exportar datos desde una tabla a un archivo en el servidor. Es útil para extraer datos y prepararlos para su carga en otros sistemas.
Sintaxis:

Listing 1: SELECT INTO FILE

```
SELECT * INTO OUTFILE 'ruta/al/archivo.csv'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM nombre_tabla;
```

Ejemplo:

Listing 2: Exportar datos de Vehiculo

```
SELECT * INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vehiculo.csv'
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM sistemaGestionFlotillas.Vehiculo;
```

4. LOAD DATA INFILE

El comando ‘LOAD DATA INFILE‘ permite cargar datos desde un archivo en una tabla de la base de datos. Es útil para importar datos extraídos previamente.
Sintaxis:

Listing 3: LOAD DATA INFILE

```
LOAD DATA INFILE 'ruta/al/archivo.csv'
INTO TABLE nombre_tabla
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Ejemplo:

Listing 4: Cargar datos en Vehiculo

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vehiculo.csv'
INTO TABLE LCS_Principal.Vehiculo
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

5. SELECT con tablas de dos bases de datos

Es posible realizar consultas que involucren tablas de dos bases de datos diferentes utilizando la sintaxis ‘$nombre_{b}asedatos.nombre_{t}abla$‘.
Sintaxis:

Listing 5: SELECT con dos bases de datos

```
SELECT *
FROM base_datos1.tabla1
JOIN base_datos2.tabla2
ON tabla1.columna = tabla2.columna;
```

Ejemplo:

Listing 6: Consultar Vehiculo y Mantenimiento

```
SELECT
    v.vehiculoId,
    v.tipo AS tipo_vehiculo,
    v.modelo,
    v.marca,
    m.fechaServicio,
    m.tipoServicio,
    m.descripcion,
    m.costo
FROM
    `LCS-Principal`.Vehiculo v
JOIN
    `LCS2-Mantenimiento`.Mantenimiento m
ON
    v.vehiculoId = m.vehiculoId;
```

6. Esquema Conceptual Local de cada nodo

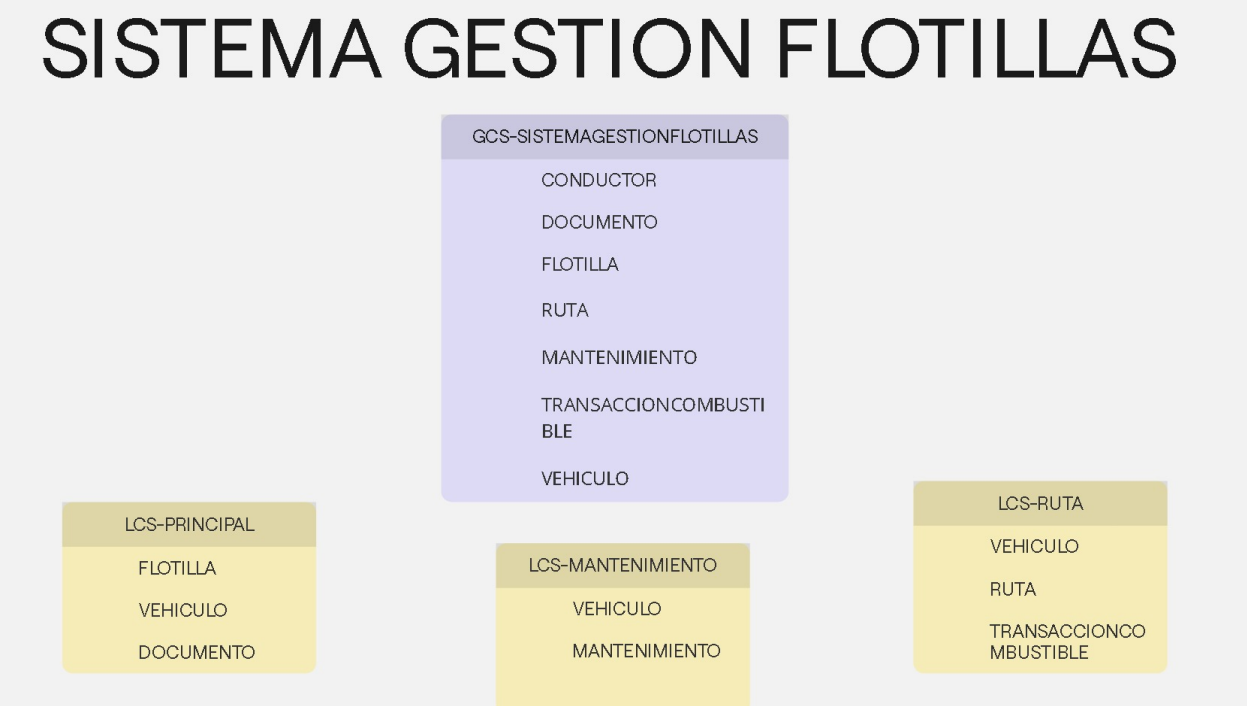

Figure 1: Concepto.

LCS-Principal:

- **Flotilla:** Almacena información sobre las flotillas.
- **Vehiculo:** Almacena información sobre los vehículos.
- **Documento:** Almacena documentos relacionados con los vehículos.

LCS2-Mantenimiento:

- **Vehiculo:** Almacena información sobre los vehículos.
- **Mantenimiento:** Almacena registros de mantenimiento de los vehículos.

LCS3-Rutas:

- **Vehiculo:** Almacena información sobre los vehículos.
- **Ruta:** Almacena información sobre las rutas realizadas por los vehículos.
- **TransaccionCombustible:** Almacena registros de transacciones de combustible.

7. Script de creación de nodos

Listing 7: Creación LCS-Principal

```
CREATE TABLE Flotilla (
    flotillaId INT NOT NULL AUTO_INCREMENT,
    nombreEmpresa VARCHAR(100) NOT NULL,
    gestorFlotilla VARCHAR(100) DEFAULT NULL,
    fechaCreacion DATE DEFAULT NULL,
    PRIMARY KEY (flotillaId)
);

CREATE TABLE Vehiculo (
    vehiculoId INT NOT NULL AUTO_INCREMENT,
    flotillaId INT NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    modelo VARCHAR(50) NOT NULL,
    marca VARCHAR(50) NOT NULL,
    anio INT NOT NULL,
    estado VARCHAR(20) DEFAULT 'Activo',
    fechaVerificacion DATE DEFAULT NULL,
    PRIMARY KEY (vehiculoId)
);

CREATE TABLE Documento (
    documentoId INT NOT NULL AUTO_INCREMENT,
    vehiculoId INT NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    fechaVencimiento DATE NOT NULL,
    estado VARCHAR(20) DEFAULT 'Vigente',
    rutaArchivo VARCHAR(255) DEFAULT NULL,
    PRIMARY KEY (documentoId)
);
```

Listing 8: Creación de LCS-Mantenimiento

```
CREATE TABLE Vehiculo (
    vehiculoId INT NOT NULL AUTO_INCREMENT,
    flotillaId INT NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    modelo VARCHAR(50) NOT NULL,
    marca VARCHAR(50) NOT NULL,
    anio INT NOT NULL,
    estado VARCHAR(20) DEFAULT 'Activo',
    fechaVerificacion DATE DEFAULT NULL,
    PRIMARY KEY (vehiculoId)
);

CREATE TABLE Mantenimiento (
    mantenimientoId INT NOT NULL AUTO_INCREMENT,
    vehiculoId INT NOT NULL,
    fechaServicio DATE NOT NULL,
    tipoServicio VARCHAR(100) NOT NULL,
    descripcion VARCHAR(200) DEFAULT NULL,
    costo DECIMAL(10,2) NOT NULL,
```

```
    estado VARCHAR(20) DEFAULT 'Completado',
    PRIMARY KEY (mantenimientoId)
);
```

Listing 9: Creación de LCS-Ruta

```
CREATE TABLE Vehiculo (
    vehiculoId INT NOT NULL AUTO_INCREMENT,
    flotillaId INT NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    modelo VARCHAR(50) NOT NULL,
    marca VARCHAR(50) NOT NULL,
    anio INT NOT NULL,
    estado VARCHAR(20) DEFAULT 'Activo',
    fechaVerificacion DATE DEFAULT NULL,
    PRIMARY KEY (vehiculoId)
);

CREATE TABLE Ruta (
    rutaId INT NOT NULL AUTO_INCREMENT,
    vehiculoId INT NOT NULL,
    conductorId INT NOT NULL,
    horaInicio DATETIME NOT NULL,
    horaFin DATETIME DEFAULT NULL,
    distancia DECIMAL(10,2) DEFAULT NULL,
    ubicacionInicio VARCHAR(100) NOT NULL,
    ubicacionFin VARCHAR(100) NOT NULL,
    estado VARCHAR(20) DEFAULT 'Pendiente',
    PRIMARY KEY (rutaId)
);

CREATE TABLE TransaccionCombustible (
    transaccionId INT NOT NULL AUTO_INCREMENT,
    vehiculoId INT NOT NULL,
    conductorId INT NOT NULL,
    monto DECIMAL(10,2) NOT NULL,
    cantidad DECIMAL(10,2) NOT NULL,
    tipoCombustible VARCHAR(20) NOT NULL,
    fechaTransaccion DATETIME NOT NULL,
    ubicacion VARCHAR(100) DEFAULT NULL,
    PRIMARY KEY (transaccionId)
);
```

8. Scripts de extracción de datos

Scripts usados para extraer datos desde GCS:

Listing 10: Flotilla

```
SELECT * FROM sistemagestionflotillas.Flotilla
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/flotilla.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 11: Vehiculo

```
SELECT * FROM sistemagestionflotillas.Vehiculo
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vehiculo.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 12: Documento

```
SELECT * FROM sistemagestionflotillas.Documento
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/documento.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 13: Mantenimiento

```
SELECT * FROM sistemagestionflotillas.Mantenimiento
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/mantenimiento.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 14: Ruta

```
SELECT * FROM sistemagestionflotillas.Ruta
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ruta.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 15: TransaccionCombustible

```
SELECT * FROM sistemagestionflotillas.TransaccionCombustible
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads
/transaccioncombustible.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

9. Script de carga de datos

Scripts para cargar datos en las bases de datos locales:
Comandos para lcs-principal:

Listing 16: Flotilla

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/flotilla.csv'
INTO TABLE `lcs-principal`.Flotilla
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 17: Vehiculo

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vehiculo.csv'
INTO TABLE `lcs-principal`.Vehiculo
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 18: Documento

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/documento.csv'
INTO TABLE `lcs-principal`.Documento
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Comandos para lcs-mantenimiento:

Listing 19: Mantenimiento

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/mantenimiento.csv'
INTO TABLE `lcs2-mantenimiento`.Mantenimiento
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 20: Vehiculo

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vehiculo.csv'
INTO TABLE `lcs2-mantenimiento`.Vehiculo
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Comandos para lcs-ruta:

Listing 21: Ruta

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ruta.csv'
INTO TABLE `lcs3-rutas`.Ruta
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 22: Vehiculo

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vehiculo.csv'
INTO TABLE `lcs3-rutas`.Vehiculo
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 23: TransaccionCombustible

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
transaccioncombustible.csv'
INTO TABLE `lcs3-rutas`.TransaccionCombustible
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

10. Script de consulta de datos a dos tablas en al menos dos nodos

Ejemplo de consultas que involucra dos nodos:

Listing 24: Consultar Vehiculo y Mantenimiento

```
SELECT
    v.vehiculoId,
    v.tipo AS tipo_vehiculo,
    v.modelo,
    v.marca,
    m.fechaServicio,
    m.tipoServicio,
    m.descripcion,
    m.costo
FROM
    `lcs-principal`.Vehiculo v
JOIN
    `lcs2-mantenimiento`.Mantenimiento m
ON
    v.vehiculoId = m.vehiculoId;
```

Listing 25: Consultar Vehiculo y Ruta

```
SELECT
    v.vehiculoId,
    v.tipo AS tipo_vehiculo,
    v.modelo,
    v.marca,
    r.rutaId,
    r.ubicacionInicio,
    r.ubicacionFin,
    r.distancia,
    tc.fechaTransaccion,
    tc.cantidad AS cantidad_combustible,
    tc.monto AS costo_combustible
FROM
    `lcs-principal`.Vehiculo v
JOIN
    `lcs3-rutas`.Ruta r
ON
    v.vehiculoId = r.vehiculoId
JOIN
    `lcs3-rutas`.TransaccionCombustible tc
ON
    r.vehiculoId = tc.vehiculoId;
```

11. Triggers

Un disparador en base de datos es un código que se ejecuta automáticamente cuando se produce un evento en una tabla. También se le conoce como trigger en inglés

Trigger 1

Listing 26: Trigger para INSERT

```
DELIMITER //

CREATE TRIGGER after_vehiculo_insert
AFTER INSERT ON `LCS-Principal`.Vehiculo
FOR EACH ROW
BEGIN
    -- Insertar en LCS2-Mantenimiento.Vehiculo
    INSERT INTO `LCS2-Mantenimiento`.Vehiculo (
        vehiculoId, flotillaId, tipo, modelo, marca, anio, estado, fechaVerificacion
    ) VALUES (
        NEW.vehiculoId, NEW.flotillaId, NEW.tipo, NEW.modelo, NEW.marca, NEW.anio, NEW
    );

    -- Insertar en LCS3-Rutas.Vehiculo
    INSERT INTO `LCS3-Rutas`.Vehiculo (
        vehiculoId, flotillaId, tipo, modelo, marca, anio, estado, fechaVerificacion
    ) VALUES (
        NEW.vehiculoId, NEW.flotillaId, NEW.tipo, NEW.modelo, NEW.marca, NEW.anio, NEW
    );
END //

DELIMITER ;
```

Trigger 2

Listing 27: Trigger para UPDATE

```
DELIMITER //

CREATE TRIGGER after_vehiculo_update
AFTER UPDATE ON `LCS-Principal`.Vehiculo
FOR EACH ROW
BEGIN
    -- Actualizar en LCS2-Mantenimiento.Vehiculo
    UPDATE `LCS2-Mantenimiento`.Vehiculo
    SET
        flotillaId = NEW.flotillaId,
        tipo = NEW.tipo,
        modelo = NEW.modelo,
        marca = NEW.marca,
        anio = NEW.anio,
        estado = NEW.estado,
        fechaVerificacion = NEW.fechaVerificacion
    WHERE vehiculoId = NEW.vehiculoId;

    -- Actualizar en LCS3-Rutas.Vehiculo
    UPDATE `LCS3-Rutas`.Vehiculo
    SET
        flotillaId = NEW.flotillaId,
```

```
        tipo = NEW.tipo,
        modelo = NEW.modelo,
        marca = NEW.marca,
        anio = NEW.anio,
        estado = NEW.estado,
        fechaVerificacion = NEW.fechaVerificacion
    WHERE vehiculoId = NEW.vehiculoId;
END //

DELIMITER ;
```

Trigger 3

Listing 28: Trigger para DELETE

```
DELIMITER //

CREATE TRIGGER after_vehiculo_delete
AFTER DELETE ON `LCS-Principal`.Vehiculo
FOR EACH ROW
BEGIN
    -- Eliminar en LCS2-Mantenimiento.Vehiculo
    DELETE FROM `LCS2-Mantenimiento`.Vehiculo
    WHERE vehiculoId = OLD.vehiculoId;

    -- Eliminar en LCS3-Rutas.Vehiculo
    DELETE FROM `LCS3-Rutas`.Vehiculo
    WHERE vehiculoId = OLD.vehiculoId;
END //

DELIMITER ;
```

12. Procedimientos Almacenados

Un procedimiento almacenado es un conjunto de instrucciones que se guardan en una base de datos y se pueden ejecutar desde otras consultas o procedimientos almacenados. Son una herramienta fundamental para la gestión de bases de datos relacionales.

Procedimiento 1

Listing 29: Procedimiento para insertar un vehículo

```
DELIMITER //
CREATE PROCEDURE InsertarVehiculo(
    IN p_flotillaId INT,
    IN p_tipo VARCHAR(50),
    IN p_modelo VARCHAR(50),
    IN p_marca VARCHAR(50),
    IN p_anio INT,
    IN p_estado VARCHAR(20),
    IN p_fechaVerificacion DATE
)
BEGIN
    -- Verificar que el a o sea v lido (por ejemplo, mayor a 2000)
    IF p_anio < 2000 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
        'El a o del veh culo debe ser mayor o igual a 2000';
```

```
    END IF;

    -- Insertar el veh culo
    INSERT INTO Vehiculo (flotillaId, tipo, modelo, marca,
    anio, estado, fechaVerificacion)
    VALUES (p_flotillaId, p_tipo, p_modelo, p_marca, p_anio, p_estado,
    p_fechaVerificacion);

    -- Mensaje de  xito
    SELECT 'Veh culo insertado correctamente' AS Mensaje;
END //

DELIMITER ;

CALL InsertarVehiculo(1, 'Pickup',
'Ford F-150', 'Ford', 2023, 'Activo', '2024-01-01');
```

Procedimiento 2

Listing 30: Procedimiento para actualizar el estado de un vehículo

```
DELIMITER //
CREATE PROCEDURE ActualizarEstadoVehiculo(
    IN p_vehiculoId INT,
    IN p_nuevoEstado VARCHAR(20)
)
BEGIN
    -- Verificar que el nuevo estado sea v lido
    IF p_nuevoEstado NOT IN ('Activo', 'Inactivo', 'Mantenimiento') THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Estado no v lido.
        Los valores permitidos son: Activo, Inactivo, Mantenimiento';
    END IF;

    -- Actualizar el estado del veh culo
    UPDATE Vehiculo
    SET estado = p_nuevoEstado
    WHERE vehiculoId = p_vehiculoId;

    -- Mensaje de  xito
    SELECT 'Estado del veh culo actualizado correctamente' AS Mensaje;
END //

DELIMITER ;
CALL ActualizarEstadoVehiculo(1, 'Inactivo');
```

Procedimiento 3

Listing 31: Procedimiento para eliminar un vehículo

```
DELIMITER //

CREATE PROCEDURE EliminarVehiculo(
    IN p_vehiculoId INT
)
BEGIN
    -- Verificar que el veh culo exista
    IF NOT EXISTS (SELECT 1 FROM Vehiculo WHERE vehiculoId
```

```
    = p_vehiculoId) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El veh culo no existe';
    END IF;

    -- Eliminar el veh culo
    DELETE FROM Vehiculo
    WHERE vehiculoId = p_vehiculoId;

    -- Mensaje de  xito
    SELECT 'Veh culo eliminado correctamente' AS Mensaje;
END //

DELIMITER ;
CALL EliminarVehiculo(1);
```

Referencias Bibliográficas

References

[1] C. Coronel, S. Morris, y P. Rob, *Database Systems: Design, Implementation, and Management*, 13ª ed. Boston, MA: Cengage Learning, 2020.

[2] R. Elmasri y S. Navathe, *Fundamentals of Database Systems*, 7ª ed. Boston, MA: Pearson, 2016.

[3] P. DuBois, *MySQL: The Complete Reference*, 2ª ed. New York, NY: McGraw-Hill Education, 2008.

[4] G. Ghiani, G. Laporte, y R. Musmanno, *Introduction to Logistics Systems Management*, 2ª ed. Chichester, UK: Wiley, 2013.

[5] M. J. Hernandez, *Database Design for Mere Mortals*, 3ª ed. Boston, MA: Addison-Wesley, 2013.

[6] M. T. Özsu y P. Valduriez, *Principles of Distributed Database Systems*, 3ª ed. New York, NY: Springer, 2011.

[7] R. Kimball y M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 2ª ed. Indianapolis, IN: Wiley, 2008.

[8] Oracle Corporation, *MySQL 8.0 Reference Manual*, 2020. [En línea]. Disponible en: https://dev.mysql.com/doc/refman/8.0/en/

[9] R. Ramakrishnan y J. Gehrke, *Database Management Systems*, 3ª ed. New York, NY: McGraw-Hill, 2003.

[10] C. J. Date, *Database Design and Relational Theory: Normal Forms and All That Jazz*, 2ª ed. Sebastopol, CA: O'Reilly Media, 2012.