Universidad Autónoma del Estado de Hidalgo

REPORTE DE PRÁCTICA NO. 2.4

NODOS

ALUMNO:Israel Campos Vázquez
Dr. Eduardo Cornejo-Velázquez

1. Introducción

En el ámbito de la gestión y manipulación de bases de datos, es fundamental comprender diversas técnicas y procesos que optimizan el almacenamiento, la consulta y la integración de datos. Este documento aborda cinco temas clave:

Fragmentación Vertical: Una estrategia de bases de datos distribuidas que mejora la eficiencia de acceso a la información mediante la división de tablas en subconjuntos de columnas.

Procesos ETL (Extract, Transform, Load): Un conjunto de procedimientos utilizados para extraer, transformar y cargar datos desde múltiples fuentes hacia un almacenamiento centralizado.

SELECT INTO FILE: Una técnica para exportar datos de una tabla a un archivo externo, facilitando la generación de reportes y respaldos.

LOAD: Un comando que permite la importación eficiente de datos desde archivos externos hacia una base de datos.

Consultas entre Tablas de Diferentes Bases de Datos: Métodos para unir información almacenada en distintas bases dentro de un mismo servidor.

2. Marco teórico

Fragmentación Vertical

La fragmentación vertical divide una tabla en subconjuntos de columnas, manteniendo en cada fragmento la clave primaria para garantizar la integridad de los datos. Se utiliza cuando diferentes aplicaciones o usuarios requieren acceder a distintas partes de la información de una misma entidad sin necesidad de cargar todas sus columnas.

Ventajas de la fragmentación vertical

1. Reduce el uso de memoria y mejora la eficiencia de consultas que solo necesitan ciertas columnas.
2. Optimiza el rendimiento al reducir la cantidad de datos transferidos por la red.
3. Permite almacenar fragmentos en distintos nodos según su uso, favoreciendo la escalabilidad.

Esta fragmentación evita cargar datos innecesarios en cada consulta, optimizando el rendimiento del sistema.

Como indican Elmasri y Navathe (2015)[1], "la fragmentación vertical mejora la eficiencia al reducir la cantidad de atributos accedidos en cada consulta, permitiendo una distribución óptima de la carga de trabajo en sistemas distribuidos".

Procesos ETL (Extract, Transform, Load)

Los procesos ETL (Extracción, Transformación y Carga) son una metodología clave en la gestión de datos, utilizada para integrar datos desde diferentes fuentes en un almacén de datos unificado.

Extracción (Extract): Se obtiene la información desde diversas fuentes.
Transformación (Transform): Se procesan y limpian los datos para adecuarlos a su destino.
Carga (Load): Se insertan los datos en la base de datos de destino.

SELECT INTO OUTFILE

El comando SELECT INTO FILE se usa para exportar datos desde una tabla hacia un archivo externo en sistemas MySQL y MariaDB. Es particularmente útil para generar reportes o respaldos en formatos como CSV.

Listing 1: Sintaxis

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

LOAD

El comando LOAD DATA INFILE permite importar datos desde un archivo externo a una tabla en MySQL o MariaDB. Se utiliza para cargar grandes volúmenes de datos de manera eficiente.

Listing 2: Sintaxis

```
LOAD DATA INFILE '/ruta/del/archivo.csv'
INTO TABLE empleados
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

SELECT con Tablas de Dos Bases de Datos

Es posible realizar consultas entre tablas ubicadas en diferentes bases de datos siempre que ambas sean accesibles desde el mismo servidor.

Listing 3: Sintaxis

```
SELECT a.id, a.nombre, b.sueldo
FROM db1.empleados a
JOIN db2.salarios b ON a.id = b.empleado_id;
```

3. Herramientas empleadas

1. DataGrip: Es un entorno de desarrollo integrado (IDE) para bases de datos creado por JetBrains. Está diseñado para ayudar a los desarrolladores y administradores de bases de datos a gestionar, consultar y optimizar sus bases de datos de manera eficiente.

2. MySQL Server: Es un sistema de gestión de bases de datos relacionales. Utiliza el lenguaje SQL para la administración y manipulación de datos. Se emplea para almacenar, organizar y gestionar información dependiendo el contexto en el que se utilice.

   Se suele utilizar para desarrollar aplicaciones web, gestionar datos de empresas, análisis de datos, entre otros.

4. Desarrollo

Creación de nodos

Listing 4: Principal

```
CREATE SCHEMA principal;
```

Listing 5: Mantenimiento

```
CREATE SCHEMA mantenimiento;
```

Listing 6: Rutas

```
CREATE SCHEMA rutas;
```

Local Conceptual Schema

Listing 7: Principal

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


CREATE TABLE Vehiculo
(
  vehiculoId INT AUTO_INCREMENT UNIQUE,
  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) NOT NULL DEFAULT 'Activo',
  fechaVerificacion DATE,
  PRIMARY KEY (vehiculoId),
  FOREIGN KEY (flotillaId) REFERENCES Flotilla(flotillaId)
);


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

Listing 8: Mantenimiento

```
CREATE TABLE Vehiculo
(
  vehiculoId INT AUTO_INCREMENT UNIQUE,
  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) NOT NULL DEFAULT 'Activo',
  fechaVerificacion DATE,
  PRIMARY KEY (vehiculoId),
  FOREIGN KEY (flotillaId) REFERENCES Principal.Flotilla(flotillaId)
);

CREATE TABLE Mantenimiento
(
  mantenimientoId INT AUTO_INCREMENT UNIQUE,
  vehiculoId INT NOT NULL,
  fechaServicio DATE NOT NULL,
  tipoServicio VARCHAR(100) NOT NULL,
  descripcion VARCHAR(200) NOT NULL,
  costo DECIMAL(10,2) NOT NULL,
  estado VARCHAR(20) NOT NULL DEFAULT 'Completado',
  PRIMARY KEY (mantenimientoId),
  FOREIGN KEY (vehiculoId) REFERENCES Vehiculo(vehiculoId)
);
```

Listing 9: Rutas

```
CREATE TABLE Conductor
(
  conductorId INT AUTO_INCREMENT UNIQUE,
  nombre VARCHAR(100) NOT NULL,
  numeroLicencia VARCHAR(50) NOT NULL,
  vencimientoLicencia DATE NOT NULL,
  estado VARCHAR(20) NOT NULL DEFAULT 'Activo',
  PRIMARY KEY (conductorId)
);

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

CREATE TABLE Vehiculo
(
  vehiculoId INT AUTO_INCREMENT UNIQUE,
```

```
  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) NOT NULL DEFAULT 'Activo',
  fechaVerificacion DATE,
  PRIMARY KEY (vehiculoId),
  FOREIGN KEY (flotillaId) REFERENCES Principal.Flotilla(flotillaId)
);


CREATE TABLE TransaccionCombustible
(
  transaccionId INT AUTO_INCREMENT UNIQUE,
  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) NOT NULL,
  PRIMARY KEY (transaccionId),
  FOREIGN KEY (vehiculoId) REFERENCES Vehiculo(vehiculoId),
  FOREIGN KEY (conductorId) REFERENCES Conductor (conductorId)
);
```

Extracción de datos

Listing 10: Select into outfile

```
SELECT *
INTO OUTFILE '/tmp/Conductor.txt'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM Conductor;

SELECT *
INTO OUTFILE '/tmp/Documento1.txt'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM Documento;

SELECT *
INTO OUTFILE '/tmp/Flotilla.txt'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM Flotilla;

SELECT *
INTO OUTFILE '/tmp/Mantenimiento1.txt'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM Mantenimiento;
```

```
SELECT *
INTO OUTFILE '/tmp/Ruta1.txt'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM Ruta;

SELECT *
INTO OUTFILE '/tmp/TransaccionCombustible.txt'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM TransaccionCombustible;

SELECT *
INTO OUTFILE '/tmp/carro.txt'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM Vehiculo;
```

Carga de datos

Listing 11: Load Principal

```
LOAD DATA INFILE '/tmp/Flotilla.txt'
INTO TABLE Flotilla
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';

LOAD DATA INFILE '/tmp/carro.txt'
INTO TABLE Vehiculo
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';

LOAD DATA INFILE '/tmp/Documento1.txt'
INTO TABLE Documento
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 12: Load Mantenimiento

```
LOAD DATA INFILE '/tmp/carro.txt'
INTO TABLE Vehiculo
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';

LOAD DATA INFILE '/tmp/Mantenimiento1.txt'
INTO TABLE Mantenimiento
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Listing 13: Load Rutas

```
LOAD DATA INFILE '/tmp/Conductor.txt'
INTO TABLE Conductor
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';

LOAD DATA INFILE '/tmp/carro.txt'
INTO TABLE Vehiculo
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';

LOAD DATA INFILE '/tmp/Ruta1.txt'
INTO TABLE Ruta
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Consultas a dos bases de datos

Listing 14: SELECTS

```
SELECT v.modelo, v.estado, v.tipo, r.ubicacionInicio, r.ubicacionFin, r.estado
FROM principal.Vehiculo v
JOIN rutas.Ruta r ON v.vehiculoId = r.vehiculoId;


SELECT v.modelo, v.estado, v.tipo, m.fechaServicio, m.tipoServicio, m.costo
FROM principal.Vehiculo v
JOIN mantenimiento.Mantenimiento m ON v.vehiculoId = m.vehiculoId;
```

TRIGGERS

Listing 15: Insertar vehiculo en nodos rutas y mantenimiento al ingresar un vehiculo nuevo en nodo principal

```
DELIMITER //
CREATE TRIGGER insertar_vehiculo
AFTER INSERT ON Vehiculo
FOR EACH ROW
BEGIN
    DECLARE vehiculoId, flotillaId, anio int;
    DECLARE tipo, modelo, marca varchar(50);
    DECLARE estado varchar(20) default 'Activo';
    DECLARE fechaVerificacion date;

    SET vehiculoId = NEW.vehiculoId;
    SET flotillaId = NEW.flotillaId;
    SET anio = NEW.anio;
    SET tipo = NEW.tipo;
    SET modelo = NEW.modelo;
    SET marca = NEW.marca;
    SET estado = NEW.estado;
    SET fechaVerificacion = NEW.fechaVerificacion;

    INSERT INTO mantenimiento.Vehiculo
    VALUES(vehiculoId,flotillaId,tipo,modelo,marca,anio,estado,fechaVerificacion);
```

```
    INSERT INTO rutas.Vehiculo
    VALUES(vehiculoId,flotillaId,tipo,modelo,marca,anio,estado,fechaVerificacion);
end //
DELIMITER ;
```

Listing 16: Actualizar nodos rutas y mantenimiento al actualizar un vehiculo en nodo principal

```
DELIMITER //
CREATE TRIGGER actualizar_vehiculo
AFTER UPDATE ON Vehiculo
FOR EACH ROW
BEGIN
    UPDATE 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 = OLD.vehiculoId;

    UPDATE 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 = OLD.vehiculoId;
END //
DELIMITER ;
```

Listing 17: Eliminar vehiculo en nodos rutas y mantenimiento al eliminar un vehiculo en nodo principal

```
DELIMITER //
CREATE TRIGGER eliminar_vehiculo
AFTER DELETE ON Vehiculo
FOR EACH ROW
BEGIN
    DELETE FROM mantenimiento.Vehiculo
    WHERE vehiculoId = OLD.vehiculoId;

    DELETE FROM rutas.Vehiculo
    WHERE vehiculoId = OLD.vehiculoId;
END //
DELIMITER ;
```

STORED PROCEDURES

Listing 18: Obtener rutas por vehículo

```
DELIMITER //
CREATE PROCEDURE ObtenerRutasPorVehiculo(IN vehiculoId INT)
BEGIN
    SELECT *
    FROM rutas.Ruta
```

```
    WHERE rutas.Ruta.vehiculoId = vehiculoId;
END //
DELIMITER ;

call ObtenerRutasPorVehiculo(2);
```

Listing 19: Contar mantenimientos por vehículo

```
DELIMITER //
CREATE PROCEDURE ContarMantenimientosPorVehiculo(IN vehiculoId INT, OUT total INT)
BEGIN
    SELECT COUNT(*) INTO total FROM mantenimiento.Mantenimiento WHERE mantenimiento.Mantenim
END //
DELIMITER ;

call ContarMantenimientosPorVehiculo(2, @total);
select @total;
```

5. Conclusiones

Al realizar esta práctica, aprendí a fragmentar una base de datos real y cargar la información en 3 nodos diferentes, cada uno con una función.

Usamos select into outfile, load data infile, usamos consultas que traían información de diferentes bases de datos y de manera adicional agregué el uso de triggers para automatizar y stored procedures.

Referencias Bibliográficas

References

[1] Elmasri, R., Navathe, S. B. (2015). Fundamentals of Database Systems (7th ed.). Pearson.

[2] Özsu, M. T., Valduriez, P. (2020). Principles of Distributed Database Systems (4th ed.). Springer.