REPORTE DE PRÁCTICA NO 2.4

NOMBRE DE LA PRÁCTICA: 3 nodos BDD Flotillas

ALUMNO:

José Eduardo Valles Aguilera

1. Introducción

Hacer 3 nodos físicos en la base de datos para la Gestión de Flotillas de las prácticas anteriores y después aplicarles el proceso ETL (Extract-Transform-Load).

2. Marco teórico

Fragmentación Vertical

Consiste en dividir una relación en subconjuntos de atributos (columnas) en lugar de filas. Cada fragmento debe incluir la clave primaria para poder reconstruir la relación original mediante un join.
Ejemplo: Si tenemos una tabla de empleados con atributos como ID, Nombre, Dirección, Salario y Departamento, podríamos dividirla en:
Empleado_Identidad (ID, Nombre, Dirección) Empleado_Salario (ID, Salario, Departamento) Ambos fragmentos mantienen la clave primaria ID para poder recomponer la relación original.

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

Proceso ETL en Bases de Datos

El proceso ETL (Extract, Transform, Load) es un enfoque ampliamente utilizado en la integración y manipulación de datos en entornos de bases de datos y data warehouses. Este proceso consta de tres etapas principales:

- Extracción (Extract): Consiste en la recopilación de datos desde múltiples fuentes, como bases de datos relacionales, archivos CSV, APIs o sistemas en la nube (Kimball & Caserta, 2011).
- Transformación (Transform): En esta fase, los datos extraídos se depuran, normalizan y enriquecen para asegurar su calidad y compatibilidad con el modelo de datos del sistema de destino (Inmon, 2020).
- Carga (Load): Los datos transformados se insertan en el almacén de datos o en bases de datos distribuidas para su posterior análisis y consulta (Vassiliadis, 2009).

El uso de herramientas ETL permite la automatización de estos procesos y garantiza la coherencia e integridad de los datos en entornos distribuidos.

Inmon, W. H. (2020). Building the Data Warehouse (5th ed.). Wiley.
Kimball, R., & Caserta, J. (2011). The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Wiley.
Vassiliadis, P. (2009). A survey of Extract–Transform–Load technology. International Journal of Data Warehousing and Mining, 5(3), 1-27. https://doi.org/10.4018/jdwm.2009070101

SELECT + INTO FILE

SELECT ... INTO OUTFILE en MySQL
El comando SELECT ... INTO OUTFILE permite exportar los resultados de una consulta a un archivo en el servidor. Se utiliza para generar archivos CSV, de texto o con formatos personalizados.

```
SELECT * FROM vehiculo
INTO OUTFILE '/var/lib/mysql-files/vehiculos.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Explicación

- INTO OUTFILE '/ruta/del/archivo': Especifica la ruta del archivo de salida.
- FIELDS TERMINATED BY ',': Indica que los campos estarán separados por comas.
- ENCLOSED BY '"': Cada campo estará encerrado entre comillas dobles.
- LINES TERMINATED BY ": Cada registro se guarda en una nueva línea.

Consideraciones

- Solo se puede ejecutar si el usuario tiene permisos de escritura en la carpeta del servidor.
- No se puede exportar a rutas fuera del directorio permitido (secure_file_priv).

MySQL Documentation. (2024). SELECT ... INTO OUTFILE Syntax. Recuperado de https://dev.mysql.com/doc/

LOAD DATA INFILE en MySQL

El comando LOAD DATA INFILE permite importar datos desde un archivo a una tabla en la base de datos.

```
LOAD DATA INFILE '/var/lib/mysql-files/vehiculos.csv'
INTO TABLE vehiculo
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Explicación

- INTO TABLE vehiculo: Indica la tabla donde se cargarán los datos.
- FIELDS TERMINATED BY ',': Define que los valores están separados por comas.
- ENCLOSED BY '"': Indica que los valores están entre comillas dobles.
- LINES TERMINATED BY ": Define el fin de línea para cada registro.

Consideraciones

- Se necesita habilitar LOCAL INFILE en MySQL (mysql –local-infile=1).
- La seguridad del servidor puede restringir la carga de archivos desde ubicaciones externas.

MySQL Documentation. (2024). LOAD DATA INFILE Syntax. Recuperado de https://dev.mysql.com/doc/

SELECT con tablas de dos bases de datos

En MySQL, se pueden realizar consultas entre bases de datos diferentes si están en el mismo servidor.

```
SELECT v.placa, c.nombre
FROM LCS1_Principal.vehiculo v
JOIN LCS3_Rutas.conductor c ON v.id_conductor = c.id_conductor;
```

Explicación

- LCS1_Principal.vehiculo: Hace referencia a la tabla vehiculo en la base de datos LCS1_Principal.
- LCS3_Rutas.conductor: Hace referencia a la tabla conductor en la base de datos LCS3_Rutas.
- JOIN: Une las dos tablas mediante la clave id_conductor.

Consideraciones

- Ambas bases de datos deben estar en el mismo servidor y el usuario debe tener permisos sobre ambas.

Elmasri, R., & Navathe, S. (2020). Fundamentals of Database Systems (7th ed.). Pearson.

3. Herramientas empleadas

1. MySQL Workbench. Es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, gestión y mantenimiento para el sistema de base de datos MySQL.

4. Desarrollo

Creación de nodos

```
        -- Nodo LCS1-Principal
CREATE DATABASE LCS1_Principal;
USE LCS1_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)
);

-- Nodo LCS2-Mantenimiento
CREATE DATABASE LCS2_Mantenimiento;
USE LCS2_Mantenimiento;

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 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)
);

-- Nodo LCS3-Rutas
CREATE DATABASE LCS3_Rutas;
USE LCS3_Rutas;

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 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 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 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 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)
);
```

Figure 1: Creación de nodos

Extracción de datos

```
USE sistemagestionflotillas;

SELECT * INTO OUTFILE '/tmp/Conductor.csv' FROM sistemagestionflotillas.conductor;
SELECT * INTO OUTFILE '/tmp/Documento.csv' FROM sistemagestionflotillas.documento;
SELECT * INTO OUTFILE '/tmp/Flotilla.csv' FROM sistemagestionflotillas.flotilla;
SELECT * INTO OUTFILE '/tmp/Mantenimiento.csv' FROM sistemagestionflotillas.mantenimiento;
SELECT * INTO OUTFILE '/tmp/Ruta.csv' FROM sistemagestionflotillas.ruta;
SELECT * INTO OUTFILE '/tmp/TransaccionCombustible.csv' FROM
sistemagestionflotillas.transaccioncombustible;
SELECT * INTO OUTFILE '/tmp/Vehiculo.csv' FROM sistemagestionflotillas.vehiculo;
```

Figure 2: Exportar datos a CSV

Carga de datos

```
USE LCS1_Principal;
USE LCS2_Mantenimiento;
USE LCS3_Rutas;

LOAD DATA INFILE '/tmp/Flotilla.csv' INTO TABLE LCS1_Principal.Flotilla;
LOAD DATA INFILE '/tmp/Vehiculo.csv' INTO TABLE LCS1_Principal.Vehiculo;
LOAD DATA INFILE '/tmp/Documento.csv' INTO TABLE LCS1_Principal.Documento;
LOAD DATA INFILE '/tmp/Flotilla.csv' INTO TABLE LCS2_Mantenimiento.Flotilla;
LOAD DATA INFILE '/tmp/Vehiculo.csv' INTO TABLE LCS2_Mantenimiento.Vehiculo;
LOAD DATA INFILE '/tmp/Mantenimiento.csv' INTO TABLE LCS2_Mantenimiento.Mantenimiento;
LOAD DATA INFILE '/tmp/Flotilla.csv' INTO TABLE LCS3_Rutas.Flotilla;
LOAD DATA INFILE '/tmp/Conductor.csv' INTO TABLE LCS3_Rutas.Conductor;
LOAD DATA INFILE '/tmp/Vehiculo.csv' INTO TABLE LCS3_Rutas.Vehiculo;
LOAD DATA INFILE '/tmp/Ruta.csv' INTO TABLE LCS3_Rutas.Ruta;
LOAD DATA INFILE '/tmp/TransaccionCombustible.csv' INTO TABLE
LCS3_Rutas.TransaccionCombustible;
```

Figure 3: Importar datos desde CSV

Consulta a dos bases de datos

```
SELECT
    v.vehiculoId,
    v.tipo,
    v.modelo,
    v.marca,
    r.rutaId,
    r.horaInicio,
    r.horaFin,
    r.distancia,
    r.ubicacionInicio,
    r.ubicacionFin
FROM LCS1_Principal.Vehiculo v
JOIN LCS3_Rutas.Ruta r ON v.vehiculoId = r.vehiculoId;
```

Figure 4: Consultar rutas asignadas a vehículos (LCS1_Principal y LCS3_Rutas)

Triggers

```
-- Nodo LCS1-Principal
USE LCS1\_Principal;

-- Trigger para verificar documentos vencidos
DELIMITER $$
CREATE TRIGGER actualizar_estado_documento
BEFORE UPDATE ON Documento
FOR EACH ROW
BEGIN
    IF NEW.fechaVencimiento < CURDATE() THEN
        SET NEW.estado = 'Vencido';
    END IF;
END $$
DELIMITER ;

-- Nodo LCS2-Mantenimiento
USE LCS2\_Mantenimiento;

-- Trigger para verificar el costo del mantenimiento
DELIMITER $$
CREATE TRIGGER validar_costo_mantenimiento
BEFORE INSERT ON Mantenimiento
FOR EACH ROW
BEGIN
    IF NEW.costo <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El costo debe ser mayor que 0';
    END IF;
END $$
DELIMITER ;
```

```
-- Nodo LCS3-Rutas
USE LCS3\_Rutas;

-- Trigger para evitar rutas con hora de fin menor que la de inicio
DELIMITER $$
CREATE TRIGGER validar_horas_ruta
BEFORE INSERT ON Ruta
FOR EACH ROW
BEGIN
    IF NEW.horaFin <= NEW.horaInicio THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La hora de fin debe ser
--------posterior a la hora de inicio';
    END IF;
END $$
DELIMITER ;

-- Trigger para verificar licencias vencidas de conductores
DELIMITER $$
CREATE TRIGGER actualizar_estado_conductor
BEFORE UPDATE ON Conductor
FOR EACH ROW
BEGIN
    IF NEW.vencimientoLicencia < CURDATE() THEN
        SET NEW.estado = 'Inactivo';
    END IF;
END $$
DELIMITER ;
```

Figure 5: Creación de triggers (disparadores)

Procesos almacenados

```
-- Nodo LCS1-Principal
USE LCS1_Principal;

-- Procedimiento para agregar un nuevo documento
DELIMITER $$
CREATE PROCEDURE AgregarDocumento(
    IN p_vehiculoId INT,
    IN p_tipo VARCHAR(50),
    IN p_fechaVencimiento DATE,
    IN p_rutaArchivo VARCHAR(255)
)
BEGIN
    INSERT INTO Documento (vehiculoId, tipo, fechaVencimiento, rutaArchivo)
    VALUES (p_vehiculoId, p_tipo, p_fechaVencimiento, p_rutaArchivo);
END $$
DELIMITER ;

-- Nodo LCS2-Mantenimiento
```

```
USE LCS2_Mantenimiento;

-- Procedimiento para registrar mantenimiento
DELIMITER $$
CREATE PROCEDURE RegistrarMantenimiento(
    IN p_vehiculoId INT,
    IN p_fechaServicio DATE,
    IN p_tipoServicio VARCHAR(100),
    IN p_descripcion VARCHAR(200),
    IN p_costo DECIMAL(10,2)
)
BEGIN
    INSERT INTO Mantenimiento (vehiculoId, fechaServicio, tipoServicio,
    descripcion, costo)
    VALUES (p_vehiculoId, p_fechaServicio, p_tipoServicio, p_descripcion, p_costo);
END $$
DELIMITER ;

-- Nodo LCS3-Rutas
USE LCS3_Rutas;

-- Procedimiento para asignar una ruta
DELIMITER $$
CREATE PROCEDURE AsignarRuta(
    IN p_vehiculoId INT,
    IN p_conductorId INT,
    IN p_horaInicio DATETIME,
    IN p_horaFin DATETIME,
    IN p_distancia DECIMAL(10,2),
    IN p_ubicacionInicio VARCHAR(100),
    IN p_ubicacionFin VARCHAR(100)
)
BEGIN
    INSERT INTO Ruta (vehiculoId, conductorId, horaInicio, horaFin,
    distancia, ubicacionInicio, ubicacionFin)
    VALUES (p_vehiculoId, p_conductorId, p_horaInicio, p_horaFin,
    p_distancia, p_ubicacionInicio, p_ubicacionFin);
END $$
DELIMITER ;

-- Procedimiento para registrar una transacci n de combustible
DELIMITER $$
CREATE PROCEDURE RegistrarTransaccionCombustible(
    IN p_vehiculoId INT,
    IN p_conductorId INT,
    IN p_monto DECIMAL(10,2),
    IN p_cantidad DECIMAL(10,2),
    IN p_tipoCombustible VARCHAR(20),
    IN p_fechaTransaccion DATETIME,
    IN p_ubicacion VARCHAR(100)
)
BEGIN
    INSERT INTO TransaccionCombustible (vehiculoId, conductorId, monto,
    cantidad, tipoCombustible, fechaTransaccion, ubicacion)
```

```
    VALUES (p_vehiculoId, p_conductorId, p_monto, p_cantidad,
    p_tipoCombustible, p_fechaTransaccion, p_ubicacion);
END $$
DELIMITER ;
```

Figure 6: Creación de procesos almacenados

5. Conclusiones

Aunque fue muy difícil, aprendí a fragmentar una base de datos en nodos, a usar SELECT INTO OUTFILE para exportar datos a un archivo temporal y LOAD DATA INFILE para importar esos mismos datos a las tablas de los nodos recién creados. Fue un proceso muy complejo, pero creo que ya lo entiendo mejor.

Referencias Bibliográficas

References

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

[2] Vassiliadis, P. (2009). A survey of Extract–Transform–Load technology. International Journal of Data Warehousing and Mining, 5(3), 1-27. https://doi.org/10.4018/jdwm.2009070101

[3] MySQL Documentation. (2024). SELECT ... INTO OUTFILE Syntax. Recuperado de https://dev.mysql.com/doc/

[4] MySQL Documentation. (2024). LOAD DATA INFILE Syntax. Recuperado de https://dev.mysql.com/doc/