3.2. SQL en Fragmentos

ALUMNO:

Joal David Sanchez Vidal

Vladimir Islas Batalla

SEMESTRE Y GRUPO: 6to 2

Introducciòn

En la actualidad, la gestión eficiente de los recursos empresariales es un factor clave para mantener la competitividad de las organizaciones. Dentro de este contexto, el control y la administración de las flotillas vehiculares se ha convertido en un elemento esencial, sobre todo para las empresas que dependen del transporte, la logística y la movilidad de personal o mercancías.
La Gestión de Flotilla de Autos permite registrar, monitorear y mantener información relacionada con los vehículos, conductores, rutas, mantenimientos y consumo de combustible, con el objetivo de optimizar los costos operativos y garantizar la disponibilidad de las unidades.
El objetivo de esta práctica es diseñar e implementar un sistema distribuido de bases de datos para la gestión de una flotilla de autos, utilizando MySQL como sistema de gestión de bases de datos relacional. Para ello, se configuraron tres nodos físicos independientes:
LCS1–Principal: encargado del control administrativo de la flotilla, los vehículos y la documentación asociada.
LCS2–Mantenimiento: responsable del registro y seguimiento de los servicios realizados a cada vehículo.
LCS3–Rutas: orientado a la gestión de conductores, rutas y transacciones de combustible.
Cada nodo administra una parte específica de la información total del sistema, aplicando fragmentación vertical y horizontal según el tipo de datos y la función de cada unidad. Esta estructura permite mejorar la disponibilidad, seguridad y rendimiento del sistema al distribuir los datos y la carga de trabajo entre distintos servidores.
Además, se implementan procesos ETL (Extracción, Transformación y Carga) para transferir los datos entre nodos. Estos procesos permiten extraer la información de un nodo origen, transformarla de acuerdo con las necesidades del análisis y cargarla en los nodos destino. También se utilizan scripts SQL para crear los nodos, realizar la extracción y carga de información, y ejecutar consultas distribuidas entre las diferentes bases de datos.
En conjunto, esta práctica tiene como propósito reforzar los conocimientos sobre bases de datos distribuidas y su implementación en un entorno práctico, comprendiendo los conceptos de fragmentación, replicación e interoperabilidad entre sistemas. Con ello, se busca desarrollar habilidades para diseñar arquitecturas de información robustas y escalables, aplicables en escenarios reales donde la integridad y la disponibilidad de los datos son fundamentales.

Marco Teòrico

Fragmentación Vertical

Es una técnica que consiste en dividir una tabla de una base de datos en varias tablas más pequeñas, conservando la clave principal en cada una de ellas. Esta división se realiza en función del tipo de datos, de modo que cada fragmento contiene un subconjunto de columnas de la tabla original. Gracias a ello, algunas aplicaciones pueden acceder únicamente a un fragmento específico sin necesidad de procesar la tabla completa, Por ejemplo, un fragmento puede contener los datos del perfil de usuario, mientras que otro almacena la información de las transacciones. En el sistema de gestión de flotilla, se aplicó fragmentación vertical en la tabla Vehículo, ya que cada nodo requiere solo una parte de sus atributos:
Nodo LCS1–Principal: mantiene los datos generales (marca, modelo, año, placas, tipo, estado).
Nodo LCS2–Mantenimiento: conserva los datos relacionados con los servicios (idVehiculo, marca, modelo, año, placas) y los asocia con la información de mantenimiento.
Nodo LCS3–Rutas: almacena únicamente los datos necesarios para las rutas y las transacciones de combustible (idVehiculo, marca, modelo, placas).

Procesos ETL

Los procesos ETL (Extract, Transform, Load) son un método utilizado para la integración de datos, que consiste en extraer información de distintas fuentes, transformarla para limpiarla y adaptarla según las reglas de negocio, y finalmente cargarla en un destino final, como un almacén de datos, para su análisis. Estos

procesos se emplean para consolidar datos de múltiples orígenes, obtener información de valor, mejorar la toma de decisiones y generar reportes más precisos. En esta práctica se diseñaron procesos ETL para sincronizar la información entre los nodos del sistema:
Extract (Extracción): se obtienen los registros de vehículos y flotilla desde el nodo principal mediante consultas SELECT.
Transform (Transformación): los datos se filtran y se adaptan al formato requerido por cada nodo.
Load (Carga): los datos transformados se insertan en las tablas correspondientes de los otros nodos (por ejemplo, la tabla Vehículo en los nodos LCS2 y LCS3).

SELECT + INTO FILE

La sentencia SELECT INTO FILE en SQL permite exportar los resultados de una consulta directamente a un archivo del sistema de almacenamiento. A diferencia de SELECT INTO, que crea una nueva tabla dentro de la base de datos, la versión INTO FILE guarda los datos en un archivo externo, lo que resulta útil para tareas de respaldo, intercambio o análisis de información en otros sistemas.

```
mysql> Select * From vehiculo
    -> INTO OUTFILE '/tmp/vehiculos.cvs'
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY'"'
    -> LINES TERMINATED BY '\'
```

Figure 1: Ejecucion del codigo INSERT INTO FILE

LOAD

La instrucción LOAD DATA INFILE se utiliza para cargar datos desde un archivo de texto hacia una tabla de manera rápida y eficiente. Esta herramienta permite importar grandes volúmenes de información sin necesidad de insertar los registros uno por uno, optimizando así el rendimiento del proceso de carga. También puede hacer referencia a la utilidad mysqlslap, la cual se emplea para simular cargas de trabajo y evaluar el rendimiento del servidor MySQL, permitiendo realizar pruebas de desempeño bajo diferentes condiciones. En el contexto de esta práctica, los archivos generados previamente mediante la sentencia SELECT INTO OUTFILE se cargan en las tablas de los nodos secundarios utilizando el siguiente comando:

```
'> -- CARGAR DATOS EN LCS2-MANTENIMIENTO
'> USE LCS2_Mantenimiento;
'>
'> -- Opción 1: Cargar desde archivo CSV
'> LOAD DATA INFILE '/tmp/vehiculos_LCS2.csv'
'> INTO TABLE vehiculo
'> FIELDS TERMINATED BY ','
'> ENCLOSED BY '"'
"> LINES TERMINATED BY '\n'
"> (idVehiculo, marca, modelo, anio, placas);
">
"> -- Verificar carga
"> SELECT 'Vehículos cargados en LCS2:' as Mensaje, COUNT(*) as Total FROM vehiculo;
">
"> -- Insertar datos de mantenimiento
"> INSERT INTO mantenimiento (idVehiculo, tipo_mantenimiento, fecha, costo, descripcion, responsable) VALUES
"> (1, 'Cambio de aceite', '2024-01-15', 850.00, 'Cambio de aceite y filtro', 'Taller Central'),
"> (2, 'Rotación de llantas', '2024-02-20', 450.00, 'Rotación y balanceo', 'Taller Norte'),
"> (1, 'Alineación y balanceo', '2024-03-10', 600.00, 'Alineación de dirección', 'Taller Sur');
">
"> -- CARGAR DATOS EN LCS3-RUTAS
"> USE LCS3_Rutas;
">
"> -- Cargar vehículos desde archivo
"> LOAD DATA INFILE '/tmp/vehiculos_LCS3.csv'
"> INTO TABLE vehiculo
"> FIELDS TERMINATED BY ','
"> ENCLOSED BY '"'
'> LINES TERMINATED BY '\n'
```

Figure 2: Practica de la ejecucion LOAD

SELECT con tablas de dos bases de datos

El Modelo Entidad–Relación (MER) es una herramienta gráfica utilizada para diseñar y representar la estructura de una base de datos. Fue propuesto por Peter Chen en 1976 y permite visualizar de forma clara los principales componentes del sistema: las entidades, sus atributos y las relaciones que existen entre ellas. Las entidades representan objetos del mundo real, como por ejemplo un estudiante o un curso; los atributos describen sus características, como nombre o carrera; y las relaciones muestran cómo interactúan dichas entidades, por ejemplo, un estudiante que se inscribe en un curso.

Esquema Conceptual Local de cada nodo

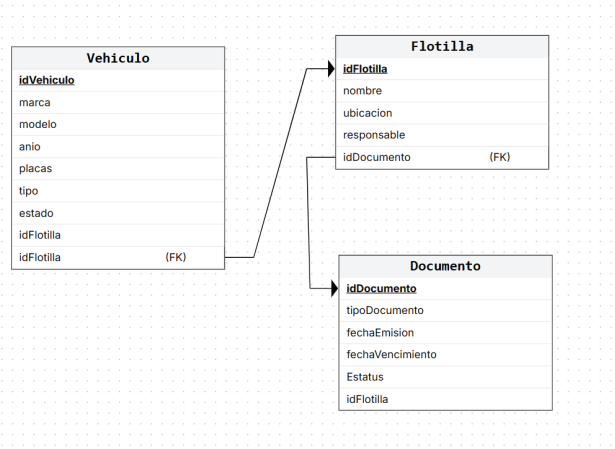

Figure 3: Nodo LCS1- Principal: flotilla, vehiculo, documento.

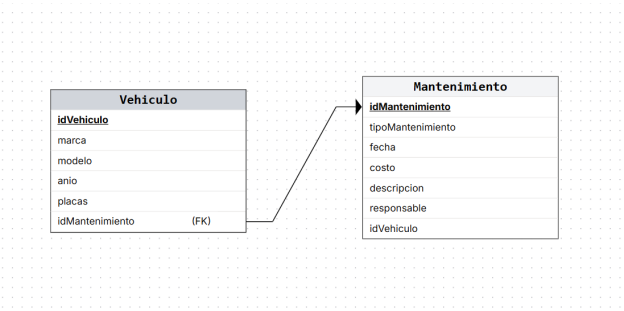

Figure 4: Nodo LCS2- Mantenimiento: vehiculo, mantenimiento

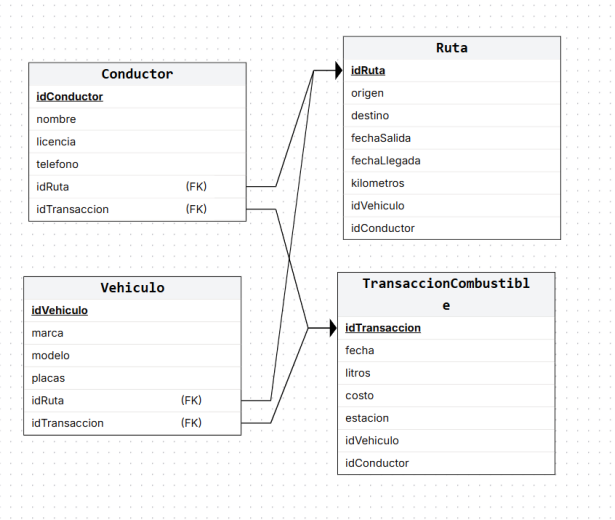

Figure 5: Nodo LCS3- Rutas: Vehiculo, conductor, ruta, transaccionCombustible.

Script de creación de nodos

Scripts SQL que crean las tablas de cada nodo (como los que ya estan para estan para LCS1, LCS2 y LCS3). Se incluyen las sentencias CREATE TABLE con las llaves foraneas y relaciones internas

```
mysql> CREATE DATABASE LCS1_Principal;
Query OK, 1 row affected (1.295 sec)

mysql> USE LCS1_Principal;
Database changed
mysql>
mysql> -- 2. Crear tabla flotilla
Query OK, 0 rows affected (0.004 sec)

mysql> CREATE TABLE flotilla(
    ->     idFlotilla INT PRIMARY KEY AUTO_INCREMENT,
    ->     nombre VARCHAR(100),
    ->     descripcion TEXT,
    ->     fecha_creacion DATE
    -> );
Query OK, 0 rows affected (1.053 sec)

mysql>
mysql> -- 3. Crear tabla vehiculo
Query OK, 0 rows affected (0.003 sec)

mysql> CREATE TABLE vehiculo(
    ->     idVehiculo INT PRIMARY KEY,
    ->     marca VARCHAR(30),
    ->     modelo VARCHAR(30),
    ->     anio INT,
    ->     placas VARCHAR(15),
    ->     tipo VARCHAR(50),
    ->     estado VARCHAR(20),
    ->     idFlotilla INT,
    ->     FOREIGN KEY (idFlotilla) REFERENCES flotilla(idFlotilla)
    -> );
Query OK, 0 rows affected (0.740 sec)

mysql>
mysql> -- 4. Crear tabla documento
Query OK, 0 rows affected (0.005 sec)

mysql> CREATE TABLE documento(
    ->     idDocumento INT PRIMARY KEY AUTO_INCREMENT,
    ->     idVehiculo INT,
```

Figure 6:

```
mysql> CREATE DATABASE LCS2_Mantenimiento;
Query OK, 1 row affected (0.099 sec)

mysql> USE LCS2_Mantenimiento;
Database changed
mysql>
mysql> -- 2. Crear tabla vehiculo (fragmentada)
Query OK, 0 rows affected (0.003 sec)

mysql> CREATE TABLE vehiculo(
    ->     idVehiculo INT PRIMARY KEY,
    ->     marca VARCHAR(30),
    ->     modelo VARCHAR(30),
    ->     anio INT,
    ->     placas VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.322 sec)

mysql>
mysql> -- 3. Crear tabla mantenimiento
Query OK, 0 rows affected (0.003 sec)

mysql> CREATE TABLE mantenimiento(
    ->     idMantenimiento INT PRIMARY KEY AUTO_INCREMENT,
    ->     idVehiculo INT,
    ->     tipo_mantenimiento VARCHAR(100),
    ->     fecha DATE,
    ->     costo DECIMAL(10,2),
    ->     descripcion TEXT,
    ->     responsable VARCHAR(100),
    ->     FOREIGN KEY (idVehiculo) REFERENCES vehiculo(idVehiculo)
    -> );
Query OK, 0 rows affected (0.740 sec)

mysql> -- 1. Crear base de datos rutas
Query OK, 0 rows affected (0.008 sec)
```

Figure 7:

```
mysql> -- 1. Crear base de datos rutas
Query OK, 0 rows affected (0.008 sec)

mysql> CREATE DATABASE LCS3_Rutas;
Query OK, 1 row affected (0.103 sec)

mysql> USE LCS3_Rutas;
Database changed
mysql>
mysql> -- 2. Crear tabla vehiculo (fragmentada)
Query OK, 0 rows affected (0.005 sec)

mysql> CREATE TABLE vehiculo(
    ->     idVehiculo INT PRIMARY KEY,
    ->     marca VARCHAR(30),
    ->     modelo VARCHAR(30),
    ->     placas VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.296 sec)

mysql>
mysql> -- 3. Crear tabla conductor
Query OK, 0 rows affected (0.003 sec)

mysql> CREATE TABLE conductor(
    ->     idConductor INT PRIMARY KEY AUTO_INCREMENT,
    ->     nombre VARCHAR(100),
    ->     licencia VARCHAR(30) UNIQUE,
    ->     telefono VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.404 sec)

mysql>
mysql> -- 4. Crear tabla ruta
Query OK, 0 rows affected (0.005 sec)

mysql> CREATE TABLE ruta(
    ->     idRuta INT PRIMARY KEY AUTO_INCREMENT,
    ->     idVehiculo INT,
    ->     idConductor INT,
    ->     origen VARCHAR(100),
```

Figure 8:

Scripts de extracción de datos

Aqui son consultas que obtienen los datos del nodo principal

```
'>
'> SELECT idVehiculo, marca, modelo, anio, placas FROM vehiculo;
'> SELECT idFlotilla, nombre FROM flotilla;
```

Figure 9:

Script de carga de datos

Inserciones o cargas desde archivo en los otros nodos:

```
'>
'> INSERT INTO vehiculo (idVehiculo, marca, modelo, anio, placas) VALUES(1, 'Nissan', 'Versa', 2024, 'ABC123');
```

Figure 10:

Script de consulta de datos a dos tablas en al menos dos de los nodos

Se utiliza la cláusula JOIN especificando las tablas en la cláusula FROM y las condiciones de unión en la cláusula ON, usando la clave primaria de una tabla y la clave foránea de la otra. La consulta básica es SELECT columnas FROM tabla1 JOIN tabla2 ON tabla1.id = tabla2.id, donde las tablas tabla1 y tabla2 se unen por sus respectivas columnas id.

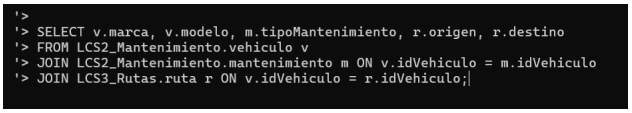

Figure 11: Descripción de la imagen

Conclusiòn

La implementación de un sistema distribuido para la Gestión de Flotilla de Autos permitió comprender, de manera práctica, cómo es posible estructurar y administrar la información de una organización a través de distintos nodos físicos y lógicos, manteniendo en todo momento la coherencia e integridad de los datos. La distribución de nodos evidenció la importancia de aplicar fragmentación vertical y horizontal para optimizar el acceso a la información, evitando la duplicación innecesaria de datos y reduciendo la carga de procesamiento concentrada en un solo servidor. Desde el punto de vista conceptual, el ejercicio reforzó los conocimientos relacionados con el diseño de sistemas de bases de datos distribuidas, la definición de esquemas conceptuales locales, el uso de claves foráneas para mantener la integridad referencial y la ejecución de consultas multibase. En el ámbito práctico, se demostró cómo una arquitectura distribuida puede mejorar la disponibilidad, seguridad y escalabilidad de un sistema de información. En conclusión, esta práctica representa una aplicación efectiva de los principios de las bases de datos distribuidas en un entorno realista. Permite observar cómo la segmentación lógica de la información y la correcta implementación de los procesos ETL contribuyen a una gestión más eficiente de los recursos, aspecto fundamental para cualquier organización que busque optimizar la administración de su flotilla y mejorar la toma de decisiones operativas.