ISSN en trámite

Caracterización de los tipos de Fragmentación en Sistemas Gestores de Bases de Datos Relacionales

Suriel Quevedo-Ortiz*, Isaac Machorro-Cano*, Lisbeth Rodríguez-Mazahua, Felipe Castro-Medina**, Mónica Guadalupe Segura-Ozuna*, Ariel López-Rodríguez***

**Universidad del Papaloapan, Tuxtepec, Oaxaca, México*
(e-mail: surielQO99@gmail.com, imachorro@unpa.edu.mx, msegura@unpa.edu.mx, alopez@unpa.edu.mx)
***Tecnológico Nacional de México/I.T. Orizaba, Orizaba, Veracruz, México*
(e-mail: lrodriguezm@ito-depi.edu.mx, dci.fcastro@ito-depi.edu.mx)

Resumen: En la actualidad, existen diversos sistemas gestores de bases de datos los cuales compiten por tener el sistema que ofrezca el mejor rendimiento en las operaciones de lectura y escritura de datos. Una forma en la que estos sistemas logran escalar una base de datos es mediante métodos de fragmentación o partición de datos. La idea básica de este concepto es dividir una tabla original en tablas más pequeñas llamadas fragmentos. Aunque actualmente los sistemas gestores de bases de datos no relacionales se empezaron a hacer populares, este trabajo se centra en los sistemas gestores de bases de datos relaciones y la caracterización de los métodos de fragmentación o partición de tablas que estos ofrecen, comparando los tiempos de respuesta en operaciones de lectura y escritura con las bases de datos estándar TPC-H y TPC-E.

Palabras clave: Caracterización, fragmentación horizontal, SGBD Relacional, TPC-H, TPC-E.

1. INTRODUCCIÓN

La fragmentación o partición es una técnica de diseño de bases de datos (BD) que consiste en dividir una tabla (original) en tablas más pequeñas (llamadas fragmentos), con el objetivo de reducir el tiempo de respuesta de las consultas. Dependiendo de los elementos (tuplas o atributos) de la tabla original que se incluyan en los fragmentos, la fragmentación es de dos tipos: horizontal, si los fragmentos contienen subconjuntos de tuplas, o vertical, si tienen subconjuntos de atributos. La mayoría de los Sistemas Gestores de Bases de Datos (SGBD) relacionales soportan distintos tipos de partición.

Ante este contexto, en este trabajo, se analizaron distintos SGBD relacionales para conocer las características que ofrecen y posteriormente, seleccionar dos para realizar una serie de pruebas de rendimiento. Para lograr lo anterior, se utilizó la base de datos estándar TPC-H y TPC-E. Esto permitió mediante distintas operaciones de lectura y escritura de datos (CRUD), obtener un estudio descriptivo para conocer las ventajas de los dos gestores con respecto a los tipos de partición que proporcionan.

Este artículo se organiza de la siguiente manera: la sección 2 presenta los trabajos relacionados con los SGBD y los tipos de fragmentación; la sección 3 presenta un análisis de los SGBD y sus métodos de fragmentación, así como la metodología de investigación utilizada para comparar los SGBD; en la sección 4 se presentan los resultados de los tiempos de respuesta obtenidos de la base de datos TPC-H y TPC-E original contra la fragmentada, así como una discusión sobre los resultados obtenidos; finalmente la sección 5 presenta las conclusiones de la investigación y el trabajo a futuro.

2. TRABAJOS RELACIONADOS

La fragmentación o partición de tablas en bases de datos, es una técnica ampliamente utilizada en la actualidad, principalmente por aquellas grandes industrias que se dedican a la recopilación de datos, ya sea en forma de texto o de archivos multimedia. A continuación, se presenta una revisión de los trabajos relacionados con los SGBD, particularmente en relación con la aplicación de técnicas de fragmentación y partición.

Qin et al. (2016) propusieron una solución de carga rápida y sin pérdida de datos, basada en herramientas de código abierto como Kafka (plataforma de transmisión de eventos distribuidos desarrollada por Apache), HDFS (*Hadoop Distributed File System*) y el SGBD Spark SQL. En consecuencia, diseñaron e implementaron un método de partición basado en la fibra de entidades, así como un algoritmo de carga en paralelo. Por otra parte, Kumar (2016) presentó estimaciones de rendimiento de las técnicas de recuperación de datos más utilizadas, como las particiones de MySQL, el particionamiento y agrupamiento del SGBD Hive y el *framework* Apache Pig. Además, Sukhija et al. (2017) presentaron una herramienta de alto nivel implementada

ISSN en trámite

en Java y SQL para el particionamiento automático de los SGBD relacionales de código abierto, utilizando esquemas de partición Hash, Key y Range. Del mismo modo, Amirthalingam y Rais (2018) propusieron un nuevo medio para automatizar el particionamiento en el SGBD Hive.

Suh et al. (2018) desarrollaron un asistente para los administradores de bases de datos (DBAs), que facilita la partición multinivel calculando un esquema de partición para una carga de trabajo concreta. También, Maabreh (2018) evaluó las técnicas de partición de datos para mejorar el rendimiento de las consultas en grandes conjuntos de datos (Big Data). Otra herramienta para las bases de datos fue diseñada y desarrollada por Miller et al. (2018), llamada ParDP, que es capaz de particionar en paralelo BD relacionales, con el objetivo de mejorar el rendimiento y la escalabilidad. Además, Costa et al. (2019) evaluaron el impacto de la partición de datos y el *bucketing* (técnica de organización de datos, que divide grandes conjuntos de datos en partes más manejables conocidas como *buckets*/cubos) en sistemas basados en Hive, utilizando distintas estrategias de organización de datos.

Por otra parte, Benkrid et al. (2020) presentaron nuevas técnicas inspiradas en la Inteligencia Artificial para contribuir en el diseño físico automatizado de una BD. Para ello, se introdujo un *framework* basado en un planificador proactivo habilitado por algoritmos de optimización genética, utilizado para particionar dinámicamente un *Big DW* (*Data Warehouse*) que se ejecutó en un clúster paralelo. Adicionalmente, Mahmud et al. (2020) presentaron un estudio exhaustivo de los métodos de particionamiento y muestreo de datos en relación con el procesamiento y el análisis de Big Data. Además, Sridevi y Sharma (2020) realizaron un estudio acerca de la partición de BD y su finalidad. Del mismo modo, se destacaron algunos de los sitios Web y aplicaciones de redes sociales más populares que utilizan una gran BD. Por otro lado, Šalgová y Matiaško (2020) analizaron el particionamiento en relación con las diversas técnicas, métodos y beneficios que aporta. Del mismo modo, los autores implementaron la técnica de partición Range y List. Además, Šalgová y Matiaško (2021) presentaron el efecto del particionamiento y la indexación en el tiempo de acceso a los datos que compararon en siete escenarios diferentes con diferentes combinaciones de particiones creadas sobre tablas e índices.

Un análisis comparativo de los trabajos relacionados indicó que las técnicas de partición horizontal son las más frecuentes, siendo el método Range el más usado, seguido del Hash, List y Key. Además, la mayoría trabajó únicamente con un SGBD. Por su parte, en este trabajo se realizaron diversas pruebas con dos de los SGBD más populares de acuerdo con (*DB-Engines Ranking*, 2021) y se implementaron todos los métodos de partición que estos ofrecen. Así mismo, se utilizaron las BD estándar TPC-H y TPC-E, las cuales no fueron consideradas en los trabajos relacionados.

3. METODOLOGÍA DE INVESTIGACIÓN

La metodología utilizada consistió de tres etapas: análisis, selección e implementación. A continuación, se describen las actividades realizadas en estas etapas.

3.1 Análisis

En esta etapa se analizaron los conceptos más importantes en la elaboración de este trabajo, como la fragmentación o partición definidas anteriormente, así como el *benchmark* TPC-H (Serlin et al., 2021), que consta de ocho tablas y 22 consultas *ad hoc* (consultas sobre la marcha) orientadas al negocio y modificaciones de datos concurrentes, y TPC-E (Serlin et al., 2015), que consta de 33 tablas para el procesamiento de transacciones en línea. TPC-H y TPC-E son especificaciones estándar desarrolladas por TPC (*Transaction Processing Performance Council*, Consejo de Rendimiento del Procesamiento de Transacciones) y sirven como punto de referencia en apoyo a la toma de decisiones. Además, este análisis permitió comprender principalmente el funcionamiento de los distintos tipos de partición existentes y que basados en esta teoría los SGBD implementan distintas técnicas para llevar a cabo dicha fragmentación en las bases de datos.

3.2 Selección

Por otra parte, en esta etapa se analizaron los cinco SGBD relacionales más populares: Oracle (*Oracle Database 21c*, 2021), MySQL (*MySQL 8.0 Reference Manual*, 2021), Microsoft SQL Server (*SQL Server technical documentation*, 2021), PostgreSQL (*PostgreSQL 14.1 Documentation*, 2021) e IBM Db2 (*IBM Db2 documentation*, 2021) de acuerdo con (*DB-Engines Ranking*, 2021).

Como resultado del análisis de los SGBD relacionales, se lograron identificar dos SGBD en donde se implementaron los tipos de partición en la base de datos TPC-H y TPC-E. Primeramente, se tiene a MySQL, el cual es de código abierto y solamente se encuentra por debajo de Oracle en cuanto a los esquemas de partición que soporta. Además, también está PostgreSQL, el cual también es de código abierto y con mayor popularidad en los últimos años de acuerdo con (DB-Engines Ranking, 2021). PostgreSQL admite los esquemas de partición horizontal como el Range, Hash y List, mientras que MySQL, además de los tres anteriores, incluye también el particionamiento Key.

3.3 Implementación

Generación y llenado de la BD TPC-H

Se generó en primera instancia, el esquema que conformó la BD original y posteriormente se crearon las tablas. El *script* que genera las tablas NATION, REGION, PART, SUPPLIER, PARTSUPP, CUSTOMER, ORDERS y LINEITEM de la BD TPC-H, los proporciona el propio *benchmark* y no se realizó

ISSN en trámite

ninguna modificación en el caso de MySQL como se muestra en (Fig. 3.1). Por su parte, en PostgreSQL primero es necesario definir el esquema de partición a realizar en las tablas seleccionadas ya que no es posible modificarlas después, por lo que se creó un *script* por cada tipo de partición definiendo en cada uno las tablas a particionar. El *script* se ejecutó haciendo uso del comando "source" o "\." en MySQL, como se describe en la (Fig. 3.2) y en PostgreSQL se realizó con el comando "\i".

```
CREATE TABLE NATION  ( N_NATIONKEY  INTEGER NOT NULL,
                       N_NAME       CHAR(25) NOT NULL,
                       N_REGIONKEY  INTEGER NOT NULL,
                       N_COMMENT    VARCHAR(152));

CREATE TABLE REGION  ( R_REGIONKEY  INTEGER NOT NULL,
                       R_NAME       CHAR(25) NOT NULL,
                       R_COMMENT    VARCHAR(152));

CREATE TABLE PART  ( P_PARTKEY     INTEGER NOT NULL,
                     P_NAME        VARCHAR(55) NOT NULL,
                     P_MFGR        CHAR(25) NOT NULL,
                     P_BRAND       CHAR(10) NOT NULL,
                     P_TYPE        VARCHAR(25) NOT NULL,
                     P_SIZE        INTEGER NOT NULL,
                     P_CONTAINER   CHAR(10) NOT NULL,
                     P_RETAILPRICE DECIMAL(15,2) NOT NULL,
                     P_COMMENT     VARCHAR(23) NOT NULL );
```

Fig. 3.1. Fragmento del *script* que crea las tablas de la BD TPC-H con MySQL

```
mysql> source D:\Documentos\Tesis\TPC-H\MySQL\dss.ddl
Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.15 sec)

Query OK, 0 rows affected (0.22 sec)

Query OK, 0 rows affected (0.55 sec)

Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.06 sec)

Query OK, 0 rows affected (0.06 sec)

Query OK, 0 rows affected (0.06 sec)
```

Fig. 3.2. Creación de tablas en la BD TPC-H en MySQL

En seguida, se agregaron las relaciones que también las proporciona el *benchmark*, al cual tampoco se le hizo modificación alguna, como se presenta en la (Fig. 3.3).

```
-- For table REGION

ALTER TABLE REGION
ADD PRIMARY KEY (R_REGIONKEY);


-- For table NATION

ALTER TABLE NATION
ADD PRIMARY KEY (N_NATIONKEY);

ALTER TABLE NATION
ADD CONSTRAINT NATION_FK1 FOREIGN KEY (n_regionkey)
REFERENCES REGION (r_regionkey);
```

Fig. 3.3. Fragmento del *script* que crea las relaciones de la BD TPC-H con MySQL

Posteriormente, se procedió a realizar el llenado de la BD original haciendo uso del comando "LOAD DATA INFILE" en MySQL y en PostgreSQL se realizó con el comando "copy". En este caso, se desarrolló un pequeño *script* como se muestra en la (Fig. 3.4), que automatiza el proceso de llenado de las tablas.

```
LOAD DATA INFILE 'C:\\tpch\\region.tbl' INTO TABLE region FIELDS TERMINATED BY '|';
LOAD DATA INFILE 'C:\\tpch\\nation.tbl' INTO TABLE nation FIELDS TERMINATED BY '|';
LOAD DATA INFILE 'C:\\tpch\\customer.tbl' INTO TABLE customer FIELDS TERMINATED BY '|';
LOAD DATA INFILE 'C:\\tpch\\orders.tbl' INTO TABLE orders FIELDS TERMINATED BY '|';
LOAD DATA INFILE 'C:\\tpch\\supplier.tbl' INTO TABLE supplier FIELDS TERMINATED BY '|';
LOAD DATA INFILE 'C:\\tpch\\part.tbl' INTO TABLE part FIELDS TERMINATED BY '|';
LOAD DATA INFILE 'C:\\tpch\\partsupp.tbl' INTO TABLE partsupp FIELDS TERMINATED BY '|';
LOAD DATA INFILE 'C:\\tpch\\lineitem.tbl' INTO TABLE lineitem FIELDS TERMINATED BY '|';
```

Fig. 3.4. *Script* que carga las tuplas en la BD TPC-H con MySQL

Una vez generada la BD original, se creó un esquema por cada tipo de partición a implementar y en cada esquema se copió la BD original. Esto se logró haciendo uso del comando "mysqldumb" y "mysql", como se observa en (Fig. 3.5). Con mysqldumb se lee la BD original y haciendo uso de pipeline (tubería que permite obtener la salida de un programa directamente en otro), se manda el resultado a la BD que recibe los datos con el comando mysql. Este proceso se realizó para cada una de las BD en donde se implementaron los distintos tipos de particionamiento mencionados anteriormente.

```
C:\Users\suri_>mysqldump -u root -p5643 tpch | mysql -u root
-p5643 tpch_range
mysqldump: [Warning] Using a password on the command line int
erface can be insecure.
mysql: [Warning] Using a password on the command line interfa
ce can be insecure.

C:\Users\suri_>
```

Fig. 3.5. Copiado de la BD original TPC-H a la fragmentada con MySQL

Particionamiento de la BD TPC-H

Teniendo las bases de datos creadas, se pasó al desarrollo de cada tipo de particionamiento que admite MySQL (Range, List, Hash y Key), así como los admitidos por PostgreSQL (Range, List y Hash). Además, el particionamiento Range, Hash y Key, se realizó sobre las tablas más pobladas: PART, PARTSUPP, CUSTOMER, ORDERS y LINEITEM, a excepción del particionamiento List, el cual solamente se realizó sobre las últimas tres anteriores, debido a que fueron las más adecuadas para este tipo de particionamiento.

Por otro lado, es importante mencionar que en MySQL se encontró con el problema de que el particionamiento de tablas no se realiza sobre índices que son llave foránea, por lo tanto, se desarrolló el *script* que elimina las llaves foráneas de las tablas a las cuales se realizó el particionamiento y esto se aplicó en cada BD antes de ser particionada. Además, para el caso del particionamiento List, fue necesario eliminar también las llaves

ISSN en trámite

primarias. Por su parte, en PostgreSQL únicamente se eliminaron las llaves primarias y foráneas con el particionamieto List.

En las (Fig. 3.6, 3.7, 3.8 y 3.9), se muestra un ejemplo de cada tipo de particionamiento implementado con MySQL. Posteriormente, se ejecutó cada *script* en su respectiva BD.

```
-- For table PART(200000)
-- Se realizaron 10 particiones con 20000 tuplas cada una
ALTER TABLE PART PARTITION BY RANGE (P_PARTKEY) (
        PARTITION P1 VALUES LESS THAN (20001),
    PARTITION P2 VALUES LESS THAN (40001),
    PARTITION P3 VALUES LESS THAN (60001),
    PARTITION P4 VALUES LESS THAN (80001),
    PARTITION P5 VALUES LESS THAN (100001),
    PARTITION P6 VALUES LESS THAN (120001),
    PARTITION P7 VALUES LESS THAN (140001),
    PARTITION P8 VALUES LESS THAN (160001),
    PARTITION P9 VALUES LESS THAN (180001),
    PARTITION P10 VALUES LESS THAN MAXVALUE
);
```

Fig. 3.6. Ejemplo de particionamiento Range en la BD TPC-H con MySQL

```
-- For table PART(200000)
ALTER TABLE PART PARTITION BY HASH (P_PARTKEY) PARTITIONS 10;

-- For table PARTSUPP(800000)
ALTER TABLE PARTSUPP PARTITION BY HASH (PS_PARTKEY) PARTITIONS 8;

-- For table CUSTOMER(150000)
ALTER TABLE CUSTOMER PARTITION BY HASH (C_CUSTKEY) PARTITIONS 5;

-- For table ORDERS(1500000)
ALTER TABLE ORDERS PARTITION BY HASH (O_ORDERKEY) PARTITIONS 6;

-- For table LINEITEM(6000000)
ALTER TABLE LINEITEM PARTITION BY HASH (L_ORDERKEY) PARTITIONS 12;
```

Fig. 3.7. Ejemplo de particionamiento Hash en la BD TPC-H con MySQL

```
-- For table PART(200000)
ALTER TABLE PART PARTITION BY KEY (P_PARTKEY) PARTITIONS 10;

-- For table PARTSUPP(800000)
ALTER TABLE PARTSUPP PARTITION BY KEY (PS_PARTKEY) PARTITIONS 8;

-- For table CUSTOMER(150000)
ALTER TABLE CUSTOMER PARTITION BY KEY (C_CUSTKEY) PARTITIONS 5;

-- For table ORDERS(1500000)
ALTER TABLE ORDERS PARTITION BY KEY (O_ORDERKEY) PARTITIONS 6;

-- For table LINEITEM(6000000)
ALTER TABLE LINEITEM PARTITION BY KEY (L_ORDERKEY) PARTITIONS 12;
```

Fig. 3.8. Ejemplo de particionamiento Key en la BD TPC-H con MySQL

```
-- For table CUSTOMER(150000)
-- Se crearon particiones por region:
-- África (C1) - América (C2) - Asia (C3) - Europa (C4) - Medio Oriente (C5)
ALTER TABLE CUSTOMER PARTITION BY LIST (C_NATIONKEY) (
        PARTITION C1 VALUES IN (0,5,14,15,16),
    PARTITION C2 VALUES IN (1,2,3,17,24),
    PARTITION C3 VALUES IN (8,9,12,18,21),
    PARTITION C4 VALUES IN (6,7,19,22,23),
    PARTITION C5 VALUES IN (4,10,11,13,20)
);
```

Fig. 3.9. Ejemplo de particionamiento List en la BD TPC-H con MySQL

Generación y llenado de la BD TPC-E

Con TPC-E, se siguió una dinámica similar a lo realizado con TPC-H, tomando en cuenta algunos puntos a diferenciar. Para la creación de la BD TPC-E con PostgreSQL, se utilizó el archivo proporcionado por (Medina, 2019) que genera las tablas de TPC-E, este se modificó en dos archivos uno para generar las tablas y otro que agrega las relaciones. Además, se realizaron las modificaciones pertinentes, tales como el cambio de tipo de dato de las variables a las admitidas con PostgreSQL, y se modificó la sintaxis para agregar los índices de las tablas. Además, en PostgreSQL primero se llenaron las tablas, después se particionaron y por último se agregaron las relaciones. En MySQL, bastó con ejecutar el volcado de la BD proporcionado por (Medina, 2019).

Particionamiento de la BD TPC-E

Por su parte, para la BD TPC-E, el particionamiento Range, Hash y Key, se realizó sobre las tablas TRADE, SETTLEMENT, TRADE_HISTORY y HOLDING_HISTORY a excepción del particionamiento List, el cual únicamente se realizó sobre las últimas dos anteriores. Además, se eliminaron las llaves foráneas y primarias de la misma forma que en TPC-H. En las (Fig. 3.10, 3.11 y 3.12), se muestra un ejemplo de cada tipo de particionamiento implementado con PostgreSQL.

```
-- For table TRADE (1728000)
CREATE TABLE T1 PARTITION OF TRADE FOR VALUES FROM (MINVALUE) TO (200000000174528);
CREATE TABLE T2 PARTITION OF TRADE FOR VALUES FROM (200000000174528) TO (200000000349056);
CREATE TABLE T3 PARTITION OF TRADE FOR VALUES FROM (200000000349056) TO (200000000523584);
CREATE TABLE T4 PARTITION OF TRADE FOR VALUES FROM (200000000523584) TO (200000000698112);
CREATE TABLE T5 PARTITION OF TRADE FOR VALUES FROM (200000000698112) TO (200000000872640);
CREATE TABLE T6 PARTITION OF TRADE FOR VALUES FROM (200000000872640) TO (200000001047168);
CREATE TABLE T7 PARTITION OF TRADE FOR VALUES FROM (200000001047168) TO (200000001221696);
CREATE TABLE T8 PARTITION OF TRADE FOR VALUES FROM (200000001221696) TO (200000001396224);
CREATE TABLE T9 PARTITION OF TRADE FOR VALUES FROM (200000001396224) TO (200000001570752);
CREATE TABLE T10 PARTITION OF TRADE FOR VALUES FROM (200000001570752) TO (MAXVALUE);
```

Fig. 3.10. Ejemplo de particionamiento Range en la BD TPC-E con PostgreSQL

```
-- For table HOLDING_HISTORY (2247265)
CREATE TABLE HH1 PARTITION OF HOLDING_HISTORY FOR VALUES IN (-800,-700,-600);
CREATE TABLE HH2 PARTITION OF HOLDING_HISTORY FOR VALUES IN (-500,-400,-300);
CREATE TABLE HH3 PARTITION OF HOLDING_HISTORY FOR VALUES IN (-200,-100,0);
CREATE TABLE HH4 PARTITION OF HOLDING_HISTORY FOR VALUES IN (100,200,300);
CREATE TABLE HH5 PARTITION OF HOLDING_HISTORY FOR VALUES IN (400,500,600);
CREATE TABLE HH6 PARTITION OF HOLDING_HISTORY FOR VALUES IN (700,800);
```

Fig. 3.11. Ejemplo de particionamiento List en la BD TPC-E con PostgreSQL

```
-- For table TRADE_HISTORY (4148112)
CREATE TABLE TH1 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 0);
CREATE TABLE TH2 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 1);
CREATE TABLE TH3 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 2);
CREATE TABLE TH4 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 3);
CREATE TABLE TH5 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 4);
CREATE TABLE TH6 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 5);
CREATE TABLE TH7 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 6);
CREATE TABLE TH8 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 7);
CREATE TABLE TH9 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 8);
CREATE TABLE TH10 PARTITION OF TRADE_HISTORY FOR VALUES WITH (MODULUS 10, REMAINDER 9);
```

Fig. 3.12. Ejemplo de particionamiento Hash en la BD TPC-E con PostgreSQL

ISSN en trámite

Además, para la evaluación de TPC-H y TPC-E, se desarrolló un pequeño código en Java con NetBeans IDE el cual se conecta a los distintos SGBD con el fin de automatizar la ejecución de las 22 consultas que el *benchmark* TPC-H proporciona y las ocho consultas elaboradas para TPC-E y así obtener los tiempos de respuesta para cada consulta. Si desea conocer las consultas de TPC-H, se encuentran en (Serlin et al., 2021).

4. RESULTADOS Y DISCUSIÓN

Los resultados del experimento se obtuvieron utilizando una computadora de escritorio con Procesador AMD Ryzen™ 3 3200G with Radeon™ Vega 8 Graphics, frecuencia base 3.6GHz. Almacenamiento SSD M.2 NVME 512GB + HDD 500GB. Memoria RAM DDR4 8GB, velocidad 3000MHz. Sistema Operativo Windows 11 de 64 bits

La Tabla 4.1 muestra las consultas elaboradas para la BD TPC-E. Por otro lado, dado que el *benchmark* TPC-E realiza consultas de escritura de datos, para el caso de las operaciones de inserción se manipuló con código el identificador (*id*) de las tuplas a insertar, de tal forma que esa misma tupla insertada fue la misma que se eliminó al ejecutar la operación de eliminación.

Table 4.1. Consultas de lectura y escritura para la BD TPC-E

No. de consulta	Descripción
1	INSERT INTO trade VALUES (id, '2005-01-11', 'CMPT', 'TLB', '1', 'AGNPRA', 800.0, 20.55, 4.3000002741E10, 'Jose Citino', 20.51, 5.0, 59.07, 1996.5, '0');
2	INSERT INTO settlement VALUES (id, 'Cash Account', '2005-01-18', -2117.75);
3	SELECT TH_DTS FROM trade_history WHERE TH_DTS BETWEEN '2005-01-03' AND '2005-01-05';
4	SELECT * FROM trade_history WHERE TH_ST_ID!='SBMT' AND TH_ST_ID!='PNDG';
5	UPDATE holding_history SET HH_BEFORE_QTY=HH_BEFORE_QTY WHERE HH_H_T_ID>=200000001641638;
6	UPDATE holding_history SET HH_AFTER_QTY=HH_AFTER_QTY WHERE HH_H_T_ID<=200000000075031;
7	DELETE FROM settlement where SE_T_ID=id;
8	DELETE FROM trade where T_ID=id;

Las Tablas 4.2 y 4.3 muestran los tiempos de respuesta (en formato de segundos) de las consultas de solo lectura en TPC-H. Cada consulta se ejecutó un total de cinco veces, a excepción de la consulta 17 y 20 con PostgreSQL, las cuales tardaron un tiempo considerable en la ejecución y por tal motivo solo se ejecutaron una vez.

Table 4.2. Comparación de tiempos de respuesta en la BD TPC-H con MySQL

No. de consulta	Sin partición	Range	List	Hash	Key
1	**17.17**	18.22	21.36	17.82	17.40
2	**0.40**	4.56	0.42	7.51	0.53
3	82.12	63.26	**40.07**	94.11	85.73
4	**4.47**	5.49	16.12	7.60	7.31
5	33.01	50.49	**14.62**	84.42	74.04
6	4.00	4.37	**1.37**	7.35	4.28
7	26.06	**19.90**	22.96	33.60	29.69
8	115.90	**93.86**	8,321.72	148.55	132.14
9	144.83	270.94	**65.84**	494.56	132.16
10	27.86	181.41	**15.22**	274.26	267.42
11	7.88	**5.12**	6.91	7.23	7.08
12	**5.88**	6.25	10.80	8.68	7.52
13	258.38	**160.30**	225.53	266.84	228.90
14	12.98	815.57	**5.23**	1,162.47	1,074.17
15	8.55	9.16	**2.97**	11.92	11.62
16	5.89	**3.42**	5.53	4.83	4.75
17	**2.11**	2.28	2.45	2.67	2.65
18	**5.52**	162.97	363.99	243.97	235.68
19	3.03	**2.71**	3.09	4.02	3.78
20	12.89	65.19	**2.33**	83.08	79.23
21	**15.92**	17.93	34.28	19.95	19.19
22	0.72	**0.68**	2.14	1.01	0.88

Como se observa en la Tabla 4.2, el particionamiento Range logró mejorar los tiempos de respuesta para siete de las consultas TPC-H, siendo las consultas 7, 8, 11, 13, 16, 19 y 22 aquellas que lograron dicha mejora. Por su parte, el particionamiento List también logró una mejora en el tiempo de respuesta de las consultas 3, 5, 6, 9, 10, 14, 15 y 20, obteniendo así una mejora en 15 de las 22 consultas, es decir, se obtuvo una mejora total del 68.18% en las BD fragmentadas con respecto a la BD original.

Por otra parte, para el caso de PostgreSQL (Tabla 4.3), el particionamiento Range logró mejorar los tiempos de respuesta para tres de las consultas de TPC-H, siendo las consultas 6, 16 y 22 aquellas que lograron dicha mejora. Además, el particionamiento List logró una mejora en el tiempo de respuesta de las consultas 8, 11, 14, 15 y 19. Así mismo, el particionamiento Hash logró una mejora en el tiempo de respuesta de las consultas 12, 13, 20 y 21, obteniendo así una mejora en 12 de las 22 consultas, es decir, se obtuvo una mejora total del 54.55% en las BD fragmentadas con respecto a la BD original.

De forma similar, las Tablas 4.4 y 4.5 muestran los tiempos de respuesta de las consultas de lectura y escritura de datos en TPC-E Además, en este caso las consultas se ejecutaron 100 veces cada una, gracias al corto tiempo que tardaron en ejecutarse.

ISSN en trámite

Table 4.3. Comparación de tiempos de respuesta en la BD TPC-H con PostgreSQL

No. de consulta	Sin partición	Range	List	Hash
1	**3.36**	3.41	3.46	3.42
2	**0.47**	0.48	0.51	0.49
3	**0.73**	3.82	1281.55	4.06
4	**0.42**	2.1	2.16	2.03
5	**0.48**	3.37	3.53	3.44
6	0.61	**0.54**	0.59	0.61
7	**0.68**	4.69	4.38	4.25
8	1.03	2.01	**0.95**	2.03
9	**2.21**	41.53	2.61	42.01
10	**0.98**	2.8	1.08	2.39
11	0.25	0.3	**0.24**	0.31
12	0.9	0.85	0.89	**0.83**
13	1.16	1.04	1.08	**1.03**
14	0.62	0.63	**0.56**	0.62
15	1.3	1.15	**1.1**	1.15
16	0.65	**0.62**	0.64	0.69
17	**5116.84**	5256.93	5520.46	5196.59
18	**3.69**	9.72	48.58	9.77
19	0.88	0.85	**0.76**	0.84
20	30839.13	31456.2	33980.43	**30745.53**
21	0.98	0.96	4.22	**0.95**
22	0.68	**0.63**	0.67	0.64

Table 4.4. Comparación de tiempos de respuesta en la BD TPC-E con MySQL

No. de consulta	Sin partición	Range	List	Hash	Key
1	0.00087	0.00103	0.00106	0.00091	**0.00086**
2	**0.00031**	0.00046	0.00046	0.00048	0.00046
3	**2.86611**	3.03084	5.49348	3.00366	3.07758
4	**3.07730**	3.32959	5.86820	3.31341	3.34779
5	0.09506	**0.09412**	2.92519	0.10711	0.10437
6	0.23180	**0.22921**	3.18076	0.24685	0.24427
7	0.00054	0.00040	0.00040	**0.00029**	0.00043
8	0.00065	0.00054	0.00046	**0.00044**	0.00051

Table 4.5. Comparación de tiempos de respuesta en la BD TPC-E con PostgreSQL

No. de consulta	Sin partición	Range	List	Hash
1	**0.00111**	0.00121	0.001424	0.002188
2	**0.00052**	0.00099	0.001077	0.001184
3	**0.24385**	0.55093	0.338835	0.549119
4	**1.68750**	1.84401	2.189544	1.850861
5	0.77639	0.84029	**0.428474**	0.766166
6	2.76766	2.15018	**1.754820**	2.305706
7	0.00045	0.00041	**0.000399**	0.000444
8	0.00076	0.00147	**0.000700**	0.001378

Como se observa en la Tabla 4.4, para el caso de MySQL, el particionamiento Range logró mejorar los tiempos de respuesta para dos de las consultas de TPC-E, siendo las consultas 5 y 6 aquellas que lograron dicha mejora. Por su parte, el particionamiento Hash logró una mejora en el tiempo de respuesta de las consultas 7 y 8. Además, el particionamiento Key también logró una mejora en el tiempo de respuesta de la consulta uno, obteniendo así una mejora en 5 de las 8 consultas, es decir, se obtuvo una mejora total del 65% en las BD fragmentadas con respecto a la BD original.

Siguiendo con PostgreSQL (Tabla 4.5), es posible apreciar que el particionamiento List obtuvo una mejora en las consultas 5, 6, 7 y 8, lo que indica que se obtuvo una mejora de rendimiento total en las BD fragmentadas de sólo un 50%.

5. CONCLUSIONES

Los métodos de particionamiento son técnicas muy útiles que permiten escalar en gran medida cuando las bases de datos empiezan a crecer de forma considerable, no obstante, con base en los experimentos realizados en este trabajo, se observó que es importante analizar bien el tipo de consultas a realizar, así como los atributos a considerar en la BD. Por otra parte, en cuanto a los resultados obtenidos se observó que MySQL fue el que logró mejores resultados, no obstante, es necesario recalcar que cada SGBD tiene sus ventajas y desventajas, aunque para el caso de la fragmentación de tablas, MySQL es el que toma la ventaja ofreciendo una mejor implementación de métodos de particionamiento, sin mencionar que cuenta con una cantidad extensa de métodos de particionamiento. Además, en MySQL también es posible realizar subparticiones, haciendo óptima la BD si la fragmentación se implementa de forma adecuada.

La principal contribución de este artículo es un análisis comparativo entre los gestores MySQL y PostgreSQL considerando los distintos tipos de partición en las bases de datos TPC-H y TPC-E. Este análisis permitió encontrar aspectos relevantes para el proyecto titulado "Desarrollo de Nuevos Métodos de Fragmentación Dinámica para Bases de Datos Multimedia".

Finalmente, con base en los métodos de partición desarrollados en este trabajo, como trabajo a futuro se pretende llevar a cabo las mismas pruebas en un entorno de desarrollo diferente, teniendo un sistema operativo Linux como anfitrión y un clúster como máquina de pruebas, con el objetivo de comparar los tiempos de respuesta en distintos sistemas operativos y sustentar aún más los resultados obtenidos.

AGRADECIMIENTOS

Los autores agradecen el apoyo por parte del Consejo Nacional de Ciencia y Tecnología (CONACYT) y a la Secretaría de Educación Pública (SEP) por el soporte financiero a través del Fondo Sectorial de Investigación para la Educación, así como al Instituto Tecnológico de Orizaba (ITO) y a la Universidad del Papaloapan (UNPA) por el apoyo otorgado para la realización de este proyecto.

Esta investigación contribuyó al proyecto titulado "Desarrollo de Nuevos Métodos de Fragmentación Dinámica para Bases de

ISSN en trámite

Datos Multimedia" apoyado por el Fondo Sectorial de Investigación para la Educación, con clave A1-S-51808.

REFERENCIAS

Amirthalingam, T., & Rais, H. M. (2018). Automated Table Partitioner (ATAP) in Apache Hive. 2018 4th International Conference on Computer and Information Sciences (ICCOINS). DOI: https://doi.org/10.1109/iccoins.2018.8510580

Benkrid, S., Mestoui, Y., Bellatreche, L., & Ordonez, C. (2020). A Genetic Optimization Physical Planner for Big Data Warehouses. 2020 IEEE International Conference on Big Data (Big Data). DOI: https://doi.org/10.1109/bigdata50022.2020.9378196

Costa, E., Costa, C., & Santos, M. Y. (2019). Evaluating partitioning and bucketing strategies for Hive-based Big Data Warehousing systems. Journal of Big Data, 6(1). DOI: https://doi.org/10.1186/s40537-019-0196-1

DB-Engines Ranking. (2021). DB-Engines. https://db-engines.com/en/ranking

IBM Db2 documentation. (2021). IBM. https://www.ibm.com/docs/en/db2

Kumar, A. S. (2016). Performance analysis of MySQL partition, hive partition-bucketing and Apache Pig. 2016 1st India International Conference on Information Processing (IICIP). DOI: https://doi.org/10.1109/iicip.2016.7975328

Maabreh, S. K. (2018). Optimizing Database Query Performance Using Table Partitioning Techniques. 2018 International Arab Conference on Information Technology (ACIT). DOI: https://doi.org/10.1109/acit.2018.8672584

Mahmud, M. S., Huang, J. Z., Salloum, S., Emara, T. Z., & Sadatdiynov, K. (2020). A survey of data partitioning and sampling methods to support big data analysis. Big Data Mining and Analytics, 3(2), 85–101. DOI: https://doi.org/10.26599/bdma.2019.9020015

Medina, F. (2019). *Aplicación de métodos de fragmentación y replicación de datos en la nube* [Tesis de Maestría, Instituto Tecnológico de Orizaba]. Repositorio TecNM Campus Orizaba. http://repositorios.orizaba.tecnm.mx:8080/xmlui/handle/123456789/399

Miller, Z., Sukhija, N., Arora, R., & Gessinger, A. (2018). Towards a Parallel User Tool (ParDP) for Automatic Data Partitioning of Relational Databases. Proceedings of the Practice and Experience on Advanced Research Computing - PEARC '18. DOI: https://doi.org/10.1145/3219104.3229255

MySQL 8.0 Reference Manual. (2021). MySQL. https://dev.mysql.com/doc/refman/8.0/en/

Oracle Database 21c. (2021). Oracle. https://docs.oracle.com/en/database/oracle/oracle-database/21/index.html

PostgreSQL 14.1 Documentation. (2021). The PostgreSQL Global Development Group. https://www.postgresql.org/docs/current/index.html

Qin, X., Chen, Y., Jin, G., Liu, Y., Cong, Y., & Du, X. (2016). Entity Fiber Based Partitioning, No Loss Staging and Fast Loading of Log Data. 2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT). DOI: https://doi.org/10.1109/pdcat.2016.052

Šalgová, V., & Matiaško, K. (2020). Reducing Data Access Time using Table Partitioning Techniques. 18th International Conference on Emerging eLearning Technologies and Applications (ICETA), 2020, pp. 564-569. DOI: https://doi.org/10.1109/ICETA51985.2020.9379231

Šalgová, V., & Matiaško, K. (2021). The Effect of Partitioning and Indexing on Data Access Time," 2021 29th Conference of Open Innovations Association (FRUCT), 2021, pp. 301-306. DOI: https://doi.org/10.23919/FRUCT52173.2021.9435500

Serlin, O., Sawyer, T., & Gray, J. (2015). *TPC-E*. https://www.tpc.org/tpce/

Serlin, O., Sawyer, T., & Gray, J. (2021). *TPC-H*. https://www.tpc.org/tpch/

SQL Server technical documentation. (2021). Microsoft. https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

Sridevi, S.V.G., & Sharma, Y.K. (2020). An Approximative Study of Database Partitioning with Respect to Popular Social Networking Websites and Applications. In: Smys S., Bestak R., Rocha Á. (eds) Inventive Computation Technologies. ICICIT 2019. Lecture Notes in Networks and Systems, vol 98. Springer. DOI: https://doi.org/10.1007/978-3-030-33846-6_94

Suh, YK., Crolotte, A., & Kostamaa, P. (2018). MLPPI Wizard: An Automated Multi-level Partitioning Tool on Analytical Workloads. KSII Transactions on Internet and Information Systems, 12(4), 1693-1713. DOI: https://doi.org/10.3837/tiis.2018.04.016

Sukhija, N., Miller, Z., & Arora, R. (2017). A High-Level Tool for Enhancing the Performance and Scalability of Open-Source Relational Databases. Proceedings of the 9th International Conference on Management of Digital EcoSystems - MEDES '17. DOI: https://doi.org/10.1145/3167020.3167031