Universidad del Papaloapan

Campus Loma Bonita

INGENIERÍA EN COMPUTACIÓN

TESIS

Caracterización y discriminación de los tipos de fragmentación disponibles en sistemas gestores de bases de datos

QUE PARA OBTENER EL GRADO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

Suriel Quevedo Ortiz

DIRECTOR DE TESIS:

Dr. Isaac Machorro Cano

CO-DIRECTOR DE TESIS:

Dra. Lisbeth Rodríguez Mazahua

LOMA BONITA, OAX. **2022**

UNIVERSIDAD DEL PAPALOAPAN

Campus Loma Bonita

Asunto: Resultado de Revisión de Tesis

Loma Bonita Oaxaca, a 17 de noviembre de 2022

Dr. Sergio Fabián Ruiz Paz
Jefe de la Carrera de Ingeniería en Computación

Por este conducto el comité de sinodales designados para el proceso de revisión de tesis con título: ***"Caracterización y discriminación de los tipos de fragmentación disponibles en sistemas gestores de bases de datos"***, del sustentante **C. Suriel Quevedo Ortiz,** le informa que la tesis ha sido sometida a un proceso de revisión y aceptada para que se presente en examen profesional.

Sin más por el momento y agradeciendo su atención, le enviamos saludos cordiales.

A t e n t a m e n t e

Dra. Nancy Pérez Castro
Sinodal

M.C. Ariel López Rodríguez
Sinodal

c.c.p: Archivo

Universidad del Papaloapan

FECHA:	18 de Noviembre del 2022
ÁREA:	Vice-Rectoría Académica
OFICIO NÚMERO:	UNPA/VRA/225/2022
ASUNTO:	Autorización de Impresión de tesis.

C. SURIEL QUEVEDO ORTIZ
P R E S E N T E:

En base al artículo 120 del reglamento de alumnos, por medio de la presente se aprueba la impresión de la tesis titulada ***"Caracterización y discriminación de los tipos de fragmentación disponibles en sistemas gestores de bases de datos"*** así como la programación del examen profesional bajo la dirección del Dr. Isaac Machorro Cano.

Sin más por el momento aprovecho la ocasión para enviarle un cordial saludo.

Atentamente.
terra ubérrima, mens aperta
Bou Lo-tama, chi jí jú

MC. HÉCTOR LÓPEZ ARJONA
Vice-Rector Académico.

VICE-RECTORIA ACADEMICA

C.c.p. Dr. Sergio Fabián Ruiz Paz Jefe de Carrera de la Ingeniería en Computación
C.c.p. L.P. Yesenia Barrientos Arenal. Jefa del Departamento de Servicios Escolares
C.c.p. Dr. Isaac Machorro Cano Director de tesis.
C.c.p. Archivo.

Agradecimientos

En primer lugar, quiero agradecer a mi familia por todo el apoyo incondicional que me brindaron, en especial a mis padres que siempre estuvieron ahí desde el inicio de mi carrera en la Universidad del Papaloapan y quienes a pesar de las circunstancias difíciles que alguna vez pasamos, siempre me apoyaron en todo. Se que con palabras no puedo agradecerles todo lo que han hecho por mí, pero trataré de mejorar cada día de mi vida para corresponder a todo el cariño y apoyo que ustedes siempre me brindaron.

También quiero agradecer a mi director de tesis, el Dr. Isaac Machorro Cano, quien con sus conocimientos, sabiduría y experiencia me dirigió en cada una de las etapas de la elaboración de mi trabajo de tesis para alcanzar la meta de culminar satisfactoriamente mi trabajo. Así mismo, agradezco a la Dra. Lisbeth Rodríguez Mazahua, por su apoyo en la realización de este proyecto y por compartir conmigo parte de sus conocimientos y experiencia.

Agradezco también a la Universidad del Papaloapan, *Campus* Loma Bonita, por todo el apoyo y la accesibilidad brindada. Además, tengo un especial agradecimiento al profesor Ariel López Rodríguez y a la profesora Nancy Pérez Castro, por todo el apoyo incondicional brindado, los estimo y aprecio mucho. Gracias por todo el conocimiento brindado a lo largo de mi carrera y por ser excelentes profesores. Se agradece también a toda la carrera de Ingeniería en Computación y todo el equipo de profesores que la conforman, por el apoyo y conocimiento brindado a lo largo de mi carrera.

Agradezco también 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.

Índice General

Índice de Figuras IV

Índice de Tablas X

Resumen XI

Abstract XII

Introducción XIII

Capítulo 1. Antecedentes 1

1.1. Marco teórico 1

1.1.1. Base de datos 1

1.1.2. Sistema Gestor de Base de Datos 2

1.1.3. SGBD relacional 3

1.1.4. SGBD NoSQL 4

1.2. Planteamiento del problema 5

1.3. Objetivos 6

1.3.1. Objetivo general 6

1.3.2. Objetivos específicos 6

1.4. Hipótesis 6

1.5. Justificación 7

1.6. Alcance(s) y limitación(es) 8

Capítulo 2. Estado del arte 10

2.1. Fragmentación en Bases de Datos 10

2.2. Análisis comparativo del estado del arte 16

Capítulo 3. Aplicación de la metodología 22

3.1. Metodología de investigación 22

3.2. Etapa de análisis 22

3.2.1. Fragmentación o partición 22

3.2.2. Tipos de fragmentación 24
3.2.3. TPC-H 25
3.2.4. TPC-E 26
3.3. Etapa de selección 27
3.3.1. Descripción general de los SGBD 27
3.3.2. Comparación y selección de los SGBD 40
3.4. Etapa de implementación 45
3.4.1. MySQL 45
3.4.1.1. Instalación y configuraciones previas 45
3.4.1.2. Generación y llenado de la BD TPC-H original 46
3.4.1.3. Fragmentación de la BD TPC-H 50
3.4.1.4. Particionamiento Range en la BD TPC-H 51
3.4.1.5. Particionamiento List en la BD TPC-H 53
3.4.1.6. Particionamiento Hash en la BD TPC-H 54
3.4.1.7. Particionamiento Key en la BD TPC-H 56
3.4.1.8. Generación y llenado de la BD TPC-E original 57
3.4.1.9. Fragmentación de la BD TPC-E 59
3.4.1.10. Particionamiento Range en la BD TPC-E 59
3.4.1.11. Particionamiento List en la BD TPC-E 60
3.4.1.12. Particionamiento Hash en la BD TPC-E 61
3.4.1.13. Particionamiento Key en la BD TPC-E 62
3.4.2. PostgreSQL 64
3.4.2.1. Instalación y configuraciones previas 64
3.4.2.2. Generación y llenado de la BD TPC-H original 64
3.4.2.3. Fragmentación de la BD TPC-H 69
3.4.2.4. Particionamiento Range en la BD TPC-H 69
3.4.2.5. Particionamiento List en la BD TPC-H 70
3.4.2.6. Particionamiento Hash en la BD TPC-H 72
3.4.2.7. Generación y llenado de la BD TPC-E original 74
3.4.2.8. Fragmentación de la BD TPC-E 78
3.4.2.9. Particionamiento Range en la BD TPC-E 78
3.4.2.10. Particionamiento List en la BD TPC-E 79

3.4.2.11. Particionamiento Hash en la BD TPC-E 81
3.4.3. MongoDB 83
3.4.3.1. Instalación y configuraciones previas 83
3.4.3.2. Generación y llenado de la BD TPC-E 84
3.4.3.3. Clúster fragmentado en MongoDB 86
3.4.3.4. Configuración del clúster fragmentado para la BD TPC-E en MongoDB 88
3.4.3.5. Fragmentación de la BD TPC-E 94
3.5. Etapa de comparación 99
3.5.1. Operaciones en la BD TPC-H original y fragmentada 99
3.5.2. Operaciones en la BD TPC-E original y fragmentada 101
3.6. Etapa de caracterización y discriminación 104
3.6.1. Descripción de los criterios de caracterización y discriminación 104
Capítulo 4. Resultados 106
4.1. Comparación del tiempo de respuesta de las operaciones en la BD TPC-H original contra la fragmentada 106
4.2. Comparación del tiempo de respuesta de las operaciones en la BD TPC-E original contra la fragmentada 108
4.3. Análisis y discusión 108
Capítulo 5. Conclusiones 112
5.1. Conclusiones 112
5.2. Trabajo futuro 113
Anexos 114
Anexo A. Códigos para la generación de las BD y la ejecución de las pruebas 114
Anexo B. Publicación 114
Referencias 115

Índice de Figuras

Figura 1.1. Los 10 SGBDs más populares en Septiembre de 2021 _____ 7

Figura 3.1. Etapas de la metodología de investigación _____ 22

Figura 3.2. Fragmento del *script* que crea las tablas de la BD TPC-H en MySQL _____ 47

Figura 3.3. Creación de tablas en la BD TPC-H en MySQL _____ 47

Figura 3.4. Fragmento del *script* que crea las relaciones de la BD TPC-H en MySQL _____ 48

Figura 3.5. Creación de relaciones en la BD TPC-H en MySQL _____ 48

Figura 3.6. *Script* que carga las tuplas en la BD TPC-H en MySQL _____ 49

Figura 3.7. Llenado de las tablas en la BD TPC-H en MySQL _____ 49

Figura 3.8. Ejemplo de importación de la BD TPC-H original a la fragmentada en MySQL _____ 50

Figura 3.9. *Script* que elimina las llaves foráneas de la BD TPC-H en MySQL _____ 50

Figura 3.10. Ejemplo de particionamiento Range en la tabla PART de TPC-H en MySQL _____ 52

Figura 3.11. Ejecución del particionamiento Range en la BD TPC-H en MySQL _____ 52

Figura 3.12. Ejemplo de particionamiento List en la tabla CUSTOMER de TPC-H en MySQL _____ 53

Figura 3.13. Eliminación de llaves primarias en la BD TPC-H en MySQL _____ 54

Figura 3.14. Ejecución del particionamiento List en la BD TPC-H en MySQL _____ 54

Figura 3.15. Particionamiento Hash de la BD TPC-H en MySQL _____ 55

Figura 3.16. Ejecución del particionamiento Hash en la BD TPC-H en MySQL ____ 55

Figura 3.17. Particionamiento Key de la BD TPC-H en MySQL ____ 56

Figura 3.18. Ejecución del particionamiento Key en la BD TPC-H en MySQL ____ 57

Figura 3.19. Fragmento del *script* que crea la BD TPC-E en MySQL ____ 58

Figura 3.20. Creación de la BD TPC-E en MySQL ____ 58

Figura 3.21. Ejemplo de particionamiento Range en la tabla TRADE de TPC-E en MySQL ____ 59

Figura 3.22. Ejecución del particionamiento Range en la BD TPC-E en MySQL ____ 60

Figura 3.23. Ejemplo de particionamiento List en la tabla TRADE_HISTORY de TPC-E en MySQL ____ 60

Figura 3.24. Eliminación de llaves primarias en la BD TPC-E con MySQL ____ 61

Figura 3.25. Ejecución del particionamiento List en la BD TPC-E en MySQL ____ 61

Figura 3.26. Particionamiento Hash de la BD TPC-E en MySQL ____ 62

Figura 3.27. Ejecución del particionamiento Hash en la BD TPC-E en MySQL ____ 62

Figura 3.28. Particionamiento Key de la BD TPC-E en MySQL ____ 63

Figura 3.29. Ejecución del particionamiento Key en la BD TPC-E en MySQL ____ 63

Figura 3.30. Creación de las tablas en la BD TPC-H en PostgreSQL ____ 65

Figura 3.31. Creación de relaciones en la BD TPC-H en PostgreSQL ____ 65

Figura 3.32. *Script* que carga las tuplas en la BD TPC-H en PostgreSQL ____ 65

Figura 3.33. Llenado de las tablas en la BD TPC-H en PostgreSQL 66

Figura 3.34. Fragmento del *script* que crea las tablas de la BD TPC-H con particionamiento Range en PostgreSQL 66

Figura 3.35. Creación de tablas y relaciones en la BD TPC-H con particionamiento Range en PostgreSQL 67

Figura 3.36. Fragmento del *script* que crea las tablas de la BD TPC-H con particionamiento List en PostgreSQL 67

Figura 3.37. Creación de tablas y relaciones en la BD TPC-H con particionamiento List en PostgreSQL 68

Figura 3.38. Fragmento del *script* que crea las tablas de la BD TPC-H con particionamiento Hash en PostgreSQL 68

Figura 3.39. Creación de tablas y relaciones en la BD TPC-H con particionamiento Hash en PostgreSQL 68

Figura 3.40. Ejemplo de particionamiento Range en la tabla ORDERS de TPC-H en PostgreSQL 69

Figura 3.41. Ejecución del particionamiento Range en la BD TPC-H en PostgreSQL 70

Figura 3.42. Llenado de las tablas en la BD TPC-H con particionamiento Range en PostgreSQL 70

Figura 3.43. Ejemplo de particionamiento List en la tabla CUSTOMER de TPC-H en PostgreSQL 71

Figura 3.44. Ejecución del particionamiento List en la BD TPC-H en PostgreSQL 71

Figura 3.45. Llenado de las tablas en la BD TPC-H con particionamiento List en PostgreSQL 72

Figura 3.46. Ejemplo de particionamiento Hash en la tabla PARTSUPP de TPC-H en PostgreSQL 72

Figura 3.47. Ejecución del particionamiento Hash en la BD TPC-H en PostgreSQL ____ 73

Figura 3.48. Llenado de las tablas en la BD TPC-H con particionamiento Hash en PostgreSQL ____ 74

Figura 3.49. Ejemplo de definición de tabla con TPC-E en PostgreSQL ____ 74

Figura 3.50. Creación de tablas en la BD TPC-E en PostgreSQL ____ 75

Figura 3.51. *Script* que carga las tuplas en la BD TPC-E en PostgreSQL ____ 75

Figura 3.52. Llenado de las tablas en la BD TPC-E en PostgreSQL ____ 76

Figura 3.53. Creación de relaciones en la BD TPC-E en PostgreSQL ____ 76

Figura 3.54. Ejemplo de definición de tabla con particionamiento Range en la BD TPC-E con PostgreSQL ____ 76

Figura 3.55. Creación de tablas en la BD TPC-E con particionamiento Range en PostgreSQL ____ 77

Figura 3.56. Ejemplo de definición de tabla con particionamiento List en la BD TPC-E con PostgreSQL ____ 77

Figura 3.57. Creación de tablas en la BD TPC-E con particionamiento List en PostgreSQL ____ 77

Figura 3.58. Ejemplo de definición de tabla con particionamiento Hash con TPC-E en PostgreSQL ____ 77

Figura 3.59. Creación de tablas en la BD TPC-E con particionamiento Hash en PostgreSQL ____ 77

Figura 3.60. Ejemplo de particionamiento Range en la tabla TRADE de TPC-E con PostgreSQL ____ 78

Figura 3.61. Ejecución del particionamiento Range en la BD TPC-E en PostgreSQL ____ 78

Figura 3.62. Llenado de las tablas en la BD TPC-E con particionamiento Range en PostgreSQL ____ 79

Figura 3.63. Creación de relaciones en la BD TPC-E con particionamiento Range en PostgreSQL ____ 79

Figura 3.64. Ejemplo de particionamiento List en la tabla HOLDING_HISTORY de TPC-E en PostgreSQL ____ 80

Figura 3.65. Ejecución del particionamiento List en la BD TPC-E en PostgreSQL ____ 80

Figura 3.66. Llenado de las tablas en la BD TPC-E con particionamiento List en PostgreSQL ____ 81

Figura 3.67. Creación de relaciones en la BD TPC-E con particionamiento List en PostgreSQL ____ 81

Figura 3.68. Ejemplo de particionamiento Hash en la tabla TRADE_HISTORY de TPC-E en PostgreSQL ____ 82

Figura 3.69. Ejecución del particionamiento Hash en la BD TPC-E en PostgreSQL ____ 82

Figura 3.70. Llenado de las tablas en la BD TPC-E con particionamiento Hash en PostgreSQL ____ 83

Figura 3.71. Creación de relaciones en la BD TPC-E con particionamiento Hash en PostgreSQL ____ 83

Figura 3.72. Importación de la tabla TRADE a la BD TPC-E en MongoDB ____ 85

Figura 3.73. Modificación del atributo TH_DTS a la colección TRADE_HISTORY de TPC-E en MongoDB ____ 86

Figura 3.74. Archivo de configuración por defecto de Mongod ____ 87

Figura 3.75. Archivo de configuración para el primer fragmento del clúster ____ 89

Figura 3.76. Ejecución del primer fragmento ____ 90
Figura 3.77. Conexión e inicialización del primer fragmento ____ 90
Figura 3.78. Archivo de configuración para el servidor de configuración ____ 91
Figura 3.79. Ejecución del servidor de configuración ____ 91
Figura 3.80. Conexión e inicialización del archivo de configuración ____ 92
Figura 3.81. Archivo de configuración de Mongos ____ 92
Figura 3.82. Ejecución del proceso Mongos ____ 93
Figura 3.83. Adición del primer fragmento al clúster ____ 93
Figura 3.84. Configuración del clúster fragmentado con 10 fragmentos ____ 94
Figura 3.85. Habilitando fragmentación en la BD TPC-E para el particionamiento Range ____ 95
Figura 3.86. Creación de la clave de fragmento Range en la colección TRADE de TPC-E ____ 95
Figura 3.87. Fragmentación de la colección TRADE con clave de fragmento Range en la BD TPC-E ____ 96
Figura 3.88. Restauración de la BD TPC-E original a la BD con particionamiento Range ____ 96
Figura 3.89. Distribución de los datos de las colecciones en la BD tpce_range ____ 97
Figura 3.90. Distribución de los datos de las colecciones en la BD tpce_hash ____ 98
Figura 3.91. Consulta número uno en la BD TPC-H ____ 100
Figura 3.92. Archivo de salida con los tiempos de ejecución de la BD TPC-H original ____ 100
Figura 3.93. Ejemplo del archivo de salida con los tiempos de ejecución de las consultas en TPC-E ____ 103

Índice de Tablas

Tabla 2.1. Análisis comparativo de trabajos relacionados _____ 19

Tabla 3.1. Cantidad de tuplas en cada tabla de la BD TPC-H _____ 25

Tabla 3.2. Cantidad de tuplas en cada tabla de la BD TPC-E _____ 26

Tabla 3.3. Comparación de los SGBDs _____ 42

Tabla 3.4. Consultas de lectura y escritura para la BD TPC-E en MySQL y PostgreSQL _____ 102

Tabla 3.5. Consultas de lectura y escritura para la BD TPC-E en MongoDB _____ 102

Tabla 4.1. Comparación de tiempos de ejecución en la BD TPC-H con MySQL _____ 106

Tabla 4.2. Comparación de tiempos de ejecución en la BD TPC-H con PostgreSQL _____ 107

Tabla 4.3. Comparación de tiempos de ejecución en la BD TPC-E _____ 108

Tabla 4.4. Discriminación de los SGBD respecto a la partición de tablas _____ 111

Resumen

La fragmentación o partición es una técnica de diseño de bases de datos 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. Aunque la mayoría de los Sistemas Gestores de Bases de Datos (SGBD) permiten aplicar diversos tipos de partición en las bases de datos, estos difieren significativamente.

Ante este contexto, en esta tesis se realizó un estudio comparativo de los SGBD más populares para conocer los tipos de partición disponibles en los mismos, posteriormente se seleccionaron algunos gestores para implementar los distintos tipos de partición en las bases de datos estándar TPC-H y TPC-E, las cuales son especificaciones estándar desarrolladas por TPC (*Transaction Processing Performance Council*, Consejo de Rendimiento del Procesamiento de Transacciones) para conocer sus ventajas en cuanto a la reducción del tiempo de ejecución de las operaciones de gestión de datos: creación, lectura, actualización y eliminación.

Por tal motivo, se implementaron los distintos tipos de partición en PostgreSQL, MySQL y MongoDB, en las bases de datos estándar TPC-H y TPC-E. Dentro de los principales resultados, se identificó que MySQL fue el que logró mejores resultados al obtener una mejora de rendimiento en por lo menos una consulta por cada tipo de operación de gestión de datos con los métodos de partición implementados. No obstante, de forma individual la partición List de PostgreSQL, logró mejorar los tiempos de ejecución en operaciones de lectura, actualización y eliminación. Además, se identificó que, si es complejo realizar un método de partición adecuado para la base de datos, el mejor SGBD es MongoDB, ya que implementa la partición Hash para lograr una distribución uniforme de los datos en los fragmentos de forma automática.

Abstract

Fragmentation or partitioning is a database design technique that consists of splitting (original) tables into smaller ones (called fragments) in order to reduce query response time. Depending on the elements (tuples or attributes) of the original table included in the fragments, fragmentation is of two types: horizontal, if the fragments contain subsets of tuples, or vertical, if they have subsets of attributes. Although most Database Management Systems (DBMS) allows the application of multi-types partition in databases, they differ significantly.

In this context, in this thesis a comparative study of the most popular DBMS was carried out to know the partition types available in them, then some managers were selected to implement the different partition types in the standard databases (TPC-H and TPC-E), which are standard specifications developed by TPC (Transaction Processing Performance Council) to know their advantages in reducing data management operations time: creation, reading, updating and elimination.

For the experimentation stage, the different types of partitioning in PostgreSQL, MySQL and MongoDB were implemented in the standard TPC-H and TPC-E databases. Among the main results, it was identified that MySQL achieved the best results by obtaining a performance improvement in at least one query for each type of data management operation with the implemented partitioning methods.

Individually, however, PostgreSQL's List partitioning achieved improved execution times in reading, updating and elimination operations. In addition, it was identified that if it is complex to perform a suitable partitioning method for the database, the best DBMS was MongoDB since it implements Hash partitioning automatically to achieve a uniform distribution of data in the shards.

Introducción

La fragmentación o partición es una técnica de diseño de bases de datos 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) soportan distintos tipos de partición.

Esta investigación contribuyó al proyecto titulado "Desarrollo de Nuevos Métodos de Fragmentación Dinámica para Bases de Datos Multimedia" apoyado por el Fondo Sectorial de Investigación para la Educación, con clave A1-S-51808, con un estudio descriptivo que permite conocer las ventajas de los distintos gestores con respecto a los tipos de partición que proporcionan, y con base en esta información comparar los métodos desarrollados con los incorporados en los gestores.

Este trabajo se organiza de la siguiente manera: el Capítulo 1 contiene el marco teórico, el planteamiento del problema, el objetivo general y específicos, la hipótesis, la justificación, así como el alcance y limitaciones del trabajo; seguido del Capítulo 2 en donde se describen los trabajos relacionados con la tesis; después se presenta el Capítulo 3 que abarca un análisis profundo de los SGBD y sus métodos de fragmentación, así como la metodología de investigación empleada para desarrollar la presente tesis; posteriormente se encuentra el Capítulo 4 en donde se comparan los resultados de los tiempos de respuesta de la base de datos TPC-H y TPC-E original contra la fragmentada; después el Capítulo 5 presenta las conclusiones y el trabajo a futuro; finalmente, se presentan los anexos y las referencias.

Capítulo 1. Antecedentes

1.1. Marco teórico

1.1.1. Base de datos

Una base de datos (BD) se observa como una especie de contenedor que tiene toda una colección de datos almacenados de forma computarizada. Ese contenedor se manipula para insertar, obtener, eliminar o actualizar datos.

Por esta razón, Ramez y Shamkant (2007) definen una BD, como una colección de datos relacionados, tales como nombres, números de teléfono y direcciones de personas, por mencionar algunos. Estos datos se encuentran grabados en un libro de direcciones indexado o están almacenados en el disco duro de un computador mediante una aplicación como Microsoft Access o Excel. A esta colección de datos relacionados con un significado implícito, es a lo que se le conoce como BD. Adicionalmente, una BD regularmente es de cualquier tamaño y complejidad, además se generan y mantienen de forma manual o computarizada. Así mismo, una BD cuenta con propiedades, tales como:

- Una BD representa algún aspecto del mundo real.
- Una BD es una colección de datos lógicamente coherente con algún tipo de significado inherente. No es correcto denominar BD a un surtido aleatorio de datos.
- Una BD se diseña, construye y rellena con datos para un propósito específico.

Por otra parte, se indica que dentro de las BD más comunes que se encuentran en funcionamiento en la actualidad, generalmente los datos se modelan en filas y columnas dentro de las tablas, haciendo que el procesamiento y la consulta de datos sea eficiente. Así mismo, se facilitan aspectos como el acceso, la administración, la modificación y la actualización, así como el control y la organización de los datos. Además, la mayoría de las BD, utilizan un lenguaje de consulta estructurado (SQL) para escribir y consultar los datos (*Oracle México*, 2021).

1.1.2. Sistema Gestor de Base de Datos

Como se mencionó anteriormente, una BD se observa como un contenedor que almacena datos. Este contenedor se manipula para realizar ciertas operaciones. Por lo tanto, al software que se encarga de toda esta gestión y manipulación de dicho contenedor, es a lo que se le conoce como: Sistema Gestor de Bases de Datos (SGBD).

El SGBD es el software que maneja todo acceso a la BD, de manera conceptual, lo que sucede es lo siguiente:

1. Un usuario emite una petición de acceso, utilizando algún lenguaje de consulta de datos, que por lo regular es SQL.
2. El SGBD interpreta esa petición y la analiza.
3. El SGBD ejecuta las operaciones necesarias sobre la base de datos almacenada.

De acuerdo con Date (2001), algunas funciones del SGBD son las siguientes:

- **Definición de datos.**

  El SGBD tiene la capacidad de aceptar definiciones de datos en su forma fuente y convertirlas a la forma de objeto correspondiente, es decir, incluir un **procesador o compilador DDL** (*Data Definition Languaje*, Lenguaje de Definición de Datos).

- **Manipulación de datos.**

  El SGBD tiene la capacidad de manejar peticiones para recuperar, actualizar o eliminar datos existentes en la base de datos o agregar nuevos datos, es decir, incluir un **procesador o compilador DML** (*Data Manipulation Languaje*, Lenguaje de Manipulación de Datos).

- **Peticiones DML, "planeadas" y "no planeadas".**

  a) Una petición **planeada** es aquella cuya necesidad fue prevista por el DBA (*Database Administrator*, Administrador de Base de Datos), antes del momento de ejecutar la petición.

b) Por otra parte, una petición **no planeada** es una petición para la que no se previó por adelantado su necesidad, por lo que el diseño físico de la BD quizás no sea el adecuado para dicha petición.

- **Seguridad e integridad de los datos.**

  El SGBD tiene la capacidad de vigilar las peticiones del usuario y rechazar todo intento de violar las restricciones de seguridad e integridad definidas por el DBA.

- **Recuperación de datos y concurrencia.**

  El SGBD, o algún otro componente de software denominado comúnmente como **administrador de transacciones**, tiene la capacidad de imponer ciertos controles de recuperación y concurrencia.

- **Diccionario de datos.**

  El SGBD tiene la capacidad de proporcionar una función de **diccionario de datos**. Este diccionario contiene "datos acerca de los datos", llamados metadatos o descriptores; es decir, definiciones de otros objetos del sistema.

- **Rendimiento.**

  El SGBD tiene la capacidad de realizar todas las tareas antes identificadas de la manera más eficiente posible.

Básicamente la finalidad del SGBD es la de proporcionar una interfaz de usuario para el sistema de base de datos. Esta interfaz se observa como un límite en el sistema, debajo del cual todo es transparente para el usuario.

1.1.3. SGBD relacional

Un SGBD hace uso de una BD relacional la cual almacena y proporciona acceso a datos relacionados entre sí. En las BD relacionales, los datos se representan por medio de tablas, en donde cada fila de la tabla es un registro con un ID único (clave) y cada columna contiene los atributos de los datos. Adicionalmente, el nombre relacional procede del hecho de que cada registro de la BD contiene información relacionada con un tema y solo con ese tema.

Date (2001) indica que los sistemas relacionales tienen su fundamento en el modelo relacional de datos, por lo que estos sistemas se ocupan de tres aspectos importantes:

Estructura de datos.

El usuario percibe la información de la BD en forma de tablas. Bajo este contexto se definen las siguientes terminologías:

- **Tupla:** corresponde a una fila de la tabla.
- **Atributo:** se refiere a una columna de la tabla.
- **Cardinalidad:** el número de tuplas que contiene la tabla.
- **Grado:** el número de atributos que contiene la tabla.
- **Dominio:** es el conjunto de valores (INTEGER, CHAR, entre otros) que toman los distintos atributos. Junto a esta noción de tipo, se asocia la noción de los operadores (=, <, >, +, *, entre otros) válidos que se aplican a valores de ese tipo.

Integridad de los datos.

Dichas tablas satisfacen ciertas restricciones de integridad, por lo que el SGBD necesita estar informado de dichas restricciones y hacerlas cumplir.

Manipulación de los datos.

La manipulación de datos se basa principalmente en el álgebra relacional y se refiere a los operadores (restringir, proyectar, producto, unión, intersección, juntar y dividir) disponibles para la recuperación de datos.

1.1.4. SGBD NoSQL

Una BD no relacional, o comúnmente conocida como BD NoSQL permite el manejo y la manipulación de datos no estructurados y semiestructurados, a diferencia de una BD relacional en donde los datos se manejan de forma estructurada. Por ejemplo, Date (2001) menciona que en un sistema jerárquico como el IMS (*Information Management System*, Sistema Gestor de Información) de IBM (*International Business Machines Corporation*, Corporación Internacional de Máquinas Comerciales), la forma de representar los datos es mediante estructuras

de árbol, y los operadores que se tienen para manipular dichas estructuras, es mediante apuntadores que representan las rutas jerárquicas hacia arriba y hacia abajo.

Por otra parte, algunos aspectos que diferencian a un sistema NoSQL de un sistema relacional son los siguientes:

- En los sistemas no relacionales, las operaciones se encuentran generalmente en el nivel de una fila o registro a la vez, a diferencia de los relacionales, que tienen la capacidad de procesamiento de conjuntos.
- Por otra parte, en un sistema no relacional la navegación es generalmente responsabilidad del usuario, a diferencia de un sistema relacional, en el que la navegación es automática.
- En cuanto a la optimización, en un sistema no relacional, es el usuario y no el sistema quien decide qué operaciones de bajo nivel se necesitan y en qué secuencia las requiere ejecutar, a diferencia de los sistemas relacionales que de la misma manera que en el punto anterior, son de forma automática.

Adicionalmente, la Web (*Oracle México*, 2021) indica que las BD NoSQL se fueron haciendo populares a medida que las aplicaciones Web se hicieron más comunes y complejas, ya que estas BD, a menudo están altamente optimizadas para operaciones de inserción y recuperación, lo que brinda ganancias significativas en escalabilidad y rendimiento para ciertos modelos de datos.

1.2. Planteamiento del problema

Aunque la mayoría de los SGBD permiten aplicar diversos tipos de partición en las bases de datos, estos difieren significativamente, por ejemplo, PostgreSQL ofrece soporte para tres formas de partición: Range, List y Hash (*PostgreSQL*[1], 2021); mientras que MySQL, además de los tres anteriores, incluye el tipo de partición Key (*MySQL*[1], 2021). Aún y con las diferentes particiones ofrecidas por los diferentes SGBD, no se conocen las ventajas que estos proporcionan en relación a la reducción del tiempo de ejecución para realizar las operaciones de gestión de

datos como: creación, lectura, actualización y eliminación en diferentes tipos de contextos.

1.3. Objetivos

1.3.1. Objetivo general

Implementar los tipos de partición que ofrecen distintos sistemas gestores de bases de datos en una base de datos estándar para conocer sus ventajas en el rendimiento de las operaciones de gestión de datos.

1.3.2. Objetivos específicos

1. Estudiar los conceptos básicos relacionados con el proyecto (A1-S-51808) para conocer el alcance de la investigación.
2. Seleccionar los gestores de bases de datos a utilizar considerando su importancia y los tipos de partición que ofrecen.
3. Implementar los tipos de partición de los gestores de bases de datos seleccionados en la base de datos estándar TPC-H y TPC-E.
4. Comparar el tiempo de respuesta de las operaciones de gestión de datos en las dos bases de datos: TPC-H y TPC-E original, y TPC-H y TPC-E fragmentada.
5. Determinar las características de los tipos de partición disponibles en los gestores de bases de datos tomando en cuenta su uso en las operaciones de gestión de datos.

1.4. Hipótesis

La implementación de los tipos de partición disponibles en los SGBD en una base de datos estándar permitirá obtener un conjunto de características que distingan estos tipos con respecto a sus ventajas en la reducción del tiempo de ejecución de las operaciones de gestión de datos.

1.5. Justificación

La eficiencia en la ejecución de las operaciones en las bases de datos es un factor crítico para el buen funcionamiento de una aplicación. Aunque existen diversas técnicas que permiten incrementar esta eficiencia como los índices, vistas materializadas y métodos de partición, estos últimos tienen la principal ventaja de que no requieren la creación de estructuras de datos adicionales, por lo que no afectan el espacio de almacenamiento y tampoco incrementan la redundancia de datos.

Por tal motivo, es necesario realizar un estudio comparativo de los SGBD más populares para conocer los tipos de partición disponibles en los mismos. Para esto, se revisó la documentación oficial de los 10 SGBD más populares (Figura 1.1) de acuerdo con (*DB-Engines*, 2021) para conocer los tipos de partición que proporcionan, posteriormente se seleccionaron algunos gestores para implementar los distintos tipos de partición en las bases de datos estándar TPC-H y TPC-E para conocer sus ventajas en cuanto a la reducción del tiempo de ejecución de las operaciones de gestión de datos: creación, lectura, actualización y eliminación.

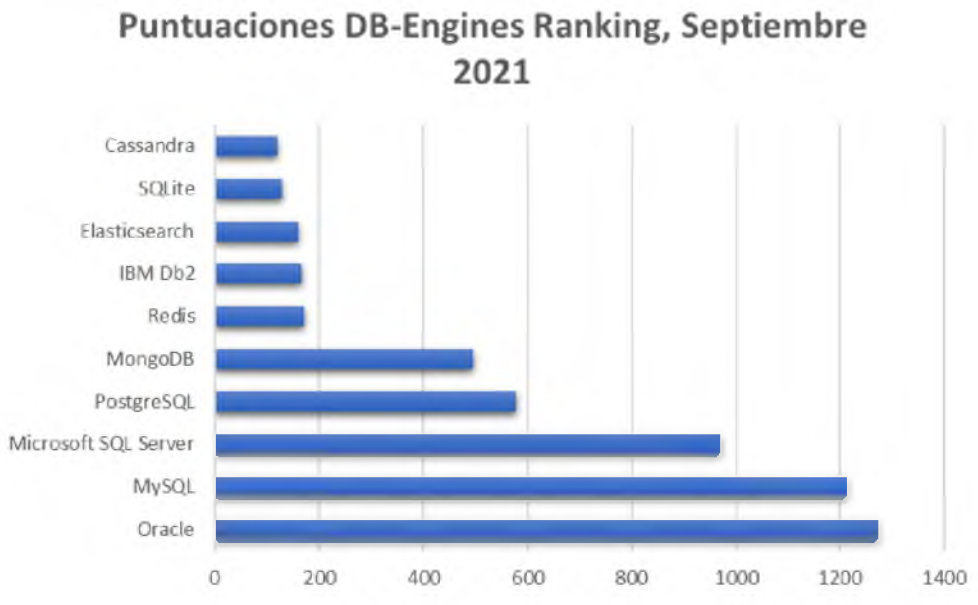


Figura 1.1. Los 10 SGBDs más populares en Septiembre de 2021

La principal contribución de esta tesis es la realización de un análisis comparativo entre los gestores relacionales MySQL y PostgreSQL, así como el gestor NoSQL MongoDB. Con este análisis se pretende identificar aspectos relevantes relacionados con los tiempos de respuesta en las consultas de gestión de datos, en base a los tipos de partición realizados sobre las bases de datos TPC-H y TPC-E.

Además, este trabajo contribuyó al proyecto denominado "Desarrollo de Nuevos Métodos de Fragmentación Dinámica para Bases de Datos Multimedia" con un estudio comparativo que permite conocer las ventajas de los distintos gestores con respecto a los tipos de partición que proporcionan, y con base en esta información comparar los métodos propuestos con los incorporados en los gestores.

1.6. Alcance(s) y limitación(es)

Dentro de los alcances y limitaciones para la implementación de los métodos de partición en las bases de datos TPC-H y TPC-E, se encuentran los siguientes:

Alcances

- Análisis de la documentación oficial de los 10 SGBD más populares y selección de los más importantes en cuanto a los tipos de partición que ofrecen los SGBD SQL y NoSQL.
- Instalación de dos SGBD relacionales y uno NoSQL, así como la implementación de los tipos de partición en las bases de datos TPC-H y TPC-E.
- Generación del código para la conexión con los SGBD seleccionados y obtención de los tiempos de respuesta de las operaciones de gestión de datos.
- Comparación de los tiempos de respuesta de las operaciones en la base de datos original contra la fragmentada y extracción de características que permitan distinguir cada tipo de partición en cuanto a las ventajas en la reducción del tiempo de ejecución de las operaciones de gestión de datos.

- Las pruebas se ejecutaron en una computadora de escritorio con sistema operativo Windows.

Limitaciones

- Una limitación de esta investigación es que sólo se contemplan 10 gestores en el análisis descriptivo.
- Además, en este trabajo sólo se implementaron los tipos de partición en MySQL, PostgreSQL y MongoDB.

Capítulo 2. Estado del arte

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 la propuesta de tesis, particularmente en relación con los SGBD y la aplicación de técnicas de fragmentación y partición. Además, se presenta un análisis comparativo de los trabajos relacionados con esta tesis.

2.1. Fragmentación en Bases de Datos

Romero et al. (2015) presentaron un estudio sobre lo viable que es resolver consultas OLAP (*On-Line Analytical Processing*, Procesamiento Analítico en Línea) con Hadoop (herramienta de Apache que implementa *MapReduce*) sobre *Big Data*, beneficiándose de los índices secundarios y el particionamiento con el SGBD HBase. Por otra parte, los autores analizaron el rendimiento de dos algoritmos, el algoritmo IRA (*Index Random Access*, Índice de Acceso Aleatorio) y el FSS (*Full Source Scan*, Escaneo Completo de la Fuente), para identificar el que presenta mejor rendimiento en términos de recursos consumidos (CPU, ancho de banda, entre otros).

Por su parte, Qin et al. (2016) propusieron una solución de carga rápida y sin pérdida de datos, basado en herramientas de código abierto como Kafka (plataforma desarrollada por Apache para manejar *Streaming Data* en tiempo real), HDFS (*Hadoop Distributed File System*, Sistema de Archivos Distribuidos de Hadoop) y el SGBD Spark SQL. En consecuencia, diseñaron e implementaron un método de partición y preparación de datos de registro basado en la fibra de entidades, así como un algoritmo de carga en paralelo. Por último, los experimentos demostraron que el esquema logró un rendimiento de preparación de datos de 390,000 registros por segundo y una carga de datos de 160,000 registros por segundo.

Así mismo, 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. Del mismo modo, en el trabajo se comprobó el tiempo de ejecución de las consultas y mediante un análisis comparativo se encontró que el tiempo de ejecución incrementa mucho debido al aumento en la cantidad de datos, especialmente con MySQL. Así mismo, se identificó que Hive es más adecuado y rápido en comparación con otros SGBD. Sin embargo, Pig resultó ser mucho más rápido incluso sin utilizar el particionamiento.

Adicionalmente, un ejemplo de las técnicas GSOP (*Generalized Skipping-Oriented Partitioning*, Particionamiento Orientado a la Omisión Generalizada) se presentó en Sun et al. (2016), en donde desarrollaron un novedoso *framework* de omisión de datos híbrido, que toma en cuenta dos componentes importantes: la agrupación de columnas y la selección local de características. Asimismo, se evaluó la eficacia de la GSOP utilizando dos referencias públicas y un *data set* (conjunto de datos) del mundo real, donde los resultados mostraron que GSOP mejoró el tiempo de respuesta con el *benchmark* TPC-H, en comparación con una versión anterior desarrollada.

Por otra parte, Sukhija et al. (2017) presentaron una herramienta de alto nivel implementada en Java y SQL para el particionamiento automático de los SGBD relacionales de código abierto, utilizando esquemas de partición por rango (*Range*), *Hash* y llave (*Key*). Así mismo, la herramienta se probó en MySQL y PostgreSQL. Del mismo modo, se evaluó el rendimiento de las consultas *Select* y *Join* después de particionar dos *data set* del mundo real: la BD USDA (*The U.S Department of Agriculture*, Departamento de Agricultura de los Estados Unidos) y la BD SDSS (*Sloan Digital Sky Survey*, Encuesta del Cielo Digital de Sloan), donde se observó un aumento significativo en el rendimiento de las consultas (hasta en un 57%).

Asimismo, Fraczek y Plechawska-Wojcik (2017) presentaron una aplicación Web desarrollada en Java y JavaScript de una red social que soporta tres tipos de BD: SQL con PostgreSQL, MongoDB y Apache Cassandra, esto con el propósito de

dar a conocer un análisis comparativo entre las BD relacionales y NoSQL. Del mismo modo, se describió el modelo de datos utilizado para cada BD y se realizaron distintas pruebas de rendimiento en el contexto de la lectura y escritura de datos. Además, dichas pruebas mostraron que MongoDB es el más rápido para leer datos y PostgreSQL el más rápido al escribir datos.

Además, Boussahoua et al. (2018) propusieron un nuevo método para la construcción de un DW (*Data Warehouse*, Almacén de Datos) distribuido, utilizando una BD NoSQL tipo columna. Del mismo modo, se utilizó el algoritmo *k-medoids*, que agrupa las consultas similares en clases, y el algoritmo PSO (*Particle Swarm Optimization*, Optimización por Enjambre de Partículas), que agrupa los atributos para crear familias de columnas según cada clase de consultas similares, para la construcción del esquema lógico. Así mismo, se evaluó dicho método utilizando el *benchmark* TPC-DS, que demostró la eficacia mediante la construcción de un almacén de datos en una BD NoSQL HBase sobre una plataforma Hadoop.

Del mismo modo, Amirthalingam y Rais (2018) propusieron un nuevo medio para automatizar el particionamiento en el SGBD Hive. Por consiguiente, se introdujo un analizador léxico que lee las consultas, para emitir a cambio, el lenguaje DDL que reestructura la tabla si una columna en particular se leyó más que el factor de coeficiente establecido por el usuario. Además, se realizaron múltiples experimentos, donde los resultados obtenidos, solidificaron en gran medida dicha propuesta de automatización.

De la misma manera, Suh et al. (2018) desarrollaron un asistente para los DBAs, que facilita la partición multinivel calculando un esquema de partición para una carga de trabajo concreta. El sistema se implementó completamente sobre un cliente, que interactúa con el subsistema de estimaciones de costos del optimizador de consultas, por medio de una API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) en la red, eliminando así, cualquier necesidad de realizar cambios en el optimizador. Además, se utilizó un algoritmo codicioso para la enumeración del espacio de búsqueda sobre los predicados de consulta en la carga de trabajo, donde obtuvieron una buena eficiencia del asistente, con una

complejidad polinómica en el peor de los casos. Por otra parte, se aplicó el asistente a la BD Teradata y se proporcionaron recomendaciones que superaron la solución de un DBA experto.

También, Maabreh (2018) evaluó las técnicas de partición de datos para mejorar el rendimiento de las consultas en *Big Data*. Para evaluar el rendimiento de las consultas, se dividió un *Big Data* en pequeños fragmentos para utilizarlos en las consultas. Así mismo, los resultados mostraron que el particionamiento mejora el rendimiento de los SGBD que gestionan bases de datos masivas, gracias a la eficacia que mostró la partición de datos en el rendimiento de las consultas. Del mismo modo, se confirmó que el tiempo medio de respuesta mejoró en un 35% en comparación con datos no particionados.

Otra herramienta para las BD 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. La herramienta proporciona una interfaz de línea de comandos desarrollada en C y SQL. Además, se realizaron pruebas en BD de MySQL y PostgreSQL, utilizando esquemas de partición Range, Hash y Key. Por otra parte, se evaluó el rendimiento de la herramienta utilizando consultas *Select* y *Join* en la BD homo_sapiens_core_92_38 del navegador *Ensembl genome*, donde se observó un rendimiento eficaz al particionar en paralelo BD heredadas en SGBD relacionales de código abierto.

Al respecto, Nam et al. (2019) explicaron los inconvenientes de los esquemas de partición basados en árboles, para lo cual, propusieron un nuevo método denominado GPT (*Graph-based Database Partitioning*, Partición de Base de Datos basado en Grafos). Asimismo, se mejoró el rendimiento de las consultas y se redujo la redundancia de datos, determinando un grafo no dirigido como esquema de partición, en lugar de un árbol. Del mismo modo, se integró el método GPT en el SGBD Spark SQL. Además, a través de extensos experimentos y con diversos puntos de referencia, como TPC-DS, IMDB (T*he Internet Movie DataBase*, La Base de Datos de Películas de Internet) Y *BioWarehouse* (BD biológica), los autores

demostraron que GPT superó significativamente al método de vanguardia en términos de sobrecarga de almacenamiento y rendimiento de la consulta.

En tal sentido, Kozma y Morschheuser (2019) presentaron una comparativa entre PostgreSQL y Apache Cassandra que se centra en el rendimiento bajo en el contexto de la latencia y las operaciones por segundo registradas por el usuario. Del mismo modo, se implementaron las operaciones estándar de BD y se utilizó la herramienta de evaluación comparativa *Yahoo! Cloud Service Benchmark* (YCSB). Además, las bases de datos se probaron utilizando una combinación de MAAS (*Metal As A Service*, servicio que permite tratar servidores físicos como máquinas virtuales en la nube) y JuJu (servicio de orquestación), ambos desarrollados por Canonical. Los resultados mostraron que Cassandra brinda un mejor rendimiento y una menor latencia en casi todas las operaciones estándar, a excepción de la escritura de datos, que tuvo un rendimiento similar en ambos casos.

Por otra parte, 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. Del mismo modo, se verificó la eficiencia en el rendimiento y con los resultados obtenidos, demostraron las ventajas de implementar *Big DWs* basados en modelos desnormalizados y el beneficio potencial de utilizar estrategias de particionamiento adecuadas. Así mismo, se encontró que las particiones alineadas con los atributos que se utilizan frecuentemente en las condiciones de las consultas aumentan significativamente la eficiencia del sistema en términos del tiempo de respuesta.

De manera similar, Khashan et al. (2020) propusieron un *framework* denominado CQNS (*Complex SQL Query and NoSQL*, Consultas Complejas SQL y NoSQL), que actúa como intérprete enviando las consultas recibidas de cualquier almacén de datos a su motor ejecutable CQNS. Por otra parte, el *framework* es capaz de soportar las consultas de la aplicación y la transformación de la BD de forma simultánea, y tiene soporte para diversas BD NoSQL, como MongoDB y Cassandra. Así mismo, se tomaron cuatro distintos escenarios de conjuntos de

datos para su evaluación en términos de optimización y tiempo de ejecución de la consulta. Los resultados mostraron que CQNS consigue una latencia y productividad óptimas en menor tiempo.

Así también, Benkrid et al. (2020) presentaron nuevas técnicas inspiradas en la IA (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* que se ejecutó en un clúster paralelo. Además, los autores encontraron que los algoritmos genéticos combinados con las técnicas en línea, son prometedores para analizar cargas de trabajo dinámicas con consultas *ad hoc* (consultas sobre la marcha), en entornos modernos de *Big Data*.

Además, 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*. Comenzaron con una visión general de los principales *frameworks* de *Big Data* en los clústeres Hadoop y discutieron los métodos clásicos de partición horizontal: Range, Hash y partición aleatoria. Del mismo modo, analizaron dos métodos habituales de muestreo de *Big Data*, el muestreo a nivel de registro y a nivel de bloque. Por último, concluyeron que el particionamiento y el muestreo de datos tienen que considerarse en conjunto para construir *frameworks* fiables tanto en el aspecto computacional como en el estadístico.

De igual forma, 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. Así mismo, el estudio mostró algunas de las bases de datos utilizadas por los sitios Web y qué tipo de esquema de partición son considerados en cada una de ellas. Además, se analizaron algunas de las características clave de los esquemas de partición de la BD de Facebook, Twitter, Amazon, WhatsApp e Instagram.

Al mismo tiempo, Šalgová y Matiaško (2020) analizaron el particionamiento en relación con las diversas técnicas, métodos y beneficios que aporta. En consecuencia, se compararon los tiempos de acceso con y sin particiones. Del

mismo modo, los autores implementaron la técnica de partición Range, que proporcionó una mejora significativa del tiempo de acceso, al igual que el particionamiento por lista (*List*), que produjo una mejora aún más significativa. Además, los resultados demostraron que las particiones, reducen significativamente el tiempo de acceso a los datos almacenados, por lo que aportan una mayor eficiencia.

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. Por tal motivo, utilizando la técnica de partición Range crearon dos tablas con datos idénticos: una sin particiones y otra con 25 particiones. Así mismo, los resultados demostraron que el uso de particiones, índices o combinaciones acelera significativamente el tiempo de acceso a los datos. No obstante, con el acceso frecuente a una gran cantidad de datos a los que se enlaza un gran número de particiones o nodos índice, el tiempo de acceso es similar al escenario sin particiones e índices.

2.2. Análisis comparativo del estado del arte

Una vez presentados y descritos los diversos trabajos relacionados sobre la partición en los SGBD, en la Tabla 2.1 se presenta un análisis comparativo, en donde se describen diversos aspectos, que fueron considerados para identificar las similitudes y particularmente las principales diferencias con los tipos de fragmentación disponibles en sistemas gestores de bases de datos que se presenta en esta tesis.

- Autor(es) – Año. - se coloca el primer autor y el año de publicación.
- Trabajo relacionado. - se realiza una descripción breve del trabajo relacionado.
- SGBD. - se indican los sistemas gestores de bases de datos utilizados.
- Tipo de fragmentación. - se indica el tipo de fragmentación utilizado.
- Dispositivo de pruebas. – se indica en qué dispositivos se realizaron las pruebas.

- Lenguaje. – se indican los lenguajes utilizados para la implementación de la BD.
- *Benchmark*. – se indican los *benchmarks* utilizados para las pruebas de rendimiento.
- Esquema de partición. – se indican los esquemas de partición utilizados.
- Tipo de consulta. – se indican los tipos de consulta utilizados en la implementación de la BD.
- *Data set*. – se indica la BD de entrada.

Además, en el último registro de la tabla se presentan los aspectos que se consideraron en esta tesis.

El resultado del análisis comparativo indica que el 45% de los trabajos relacionados utilizaron el tipo de fragmentación horizontal, un 5% utilizó la fragmentación vertical y ninguno utilizó la fragmentación híbrida. Por otra parte, PostgreSQL fue el SGBD preferido, seguido de Oracle, Cassandra, Hive, MySQL, Spark SQL y MongoDB, para la implementación de la BD. No obstante, algunos trabajos relacionados hicieron uso en menor medida, de Microsoft SQL Server, Postgres-XL, Teradata y HBase como su SGBD. Asimismo, se encontró que un 40% de los trabajos relacionados hicieron uso de algún clúster, de los cuales, un 30% utilizó un clúster físico y un 10% utilizó algún servicio que permitiera el uso de clústeres virtuales en la nube. En menor medida, se hizo uso también de PCs y *laptops*, como dispositivos de pruebas.

Por otra parte, el 75% de los trabajos relacionados optó por implementar la BD solo con lenguaje SQL, dejando de lado en su mayoría el uso de algún lenguaje de programación adicional con el cual hacer conexión con la BD. De la misma manera, solo un 45% de los trabajos relacionados hicieron uso de algún *benchmark*, como TPC-DS, TPC-H o SSB (*The Star Schema Benchmark*), para medir el rendimiento de las consultas, así mismo, solo un 20% tuvo acceso a un *data set* del mundo real.

En cuanto al esquema de partición, se observó que la mayoría utilizó los métodos clásicos de partición horizontal, siendo la partición Hash el esquema más

usado, seguido del esquema de partición Range y en menor medida el esquema de partición List y Key. Además, la mayoría de los trabajos relacionados utilizaron los tipos de operación básicos en los SGBD usando en un 70% la operación *Select*, seguido de la operación *Join* en un 40%, después la operación *Insert* en un 30%, seguido de un 25% la operación *Update* y un 20% de uso para la operación *Delete*.

Tabla 2.1. Análisis comparativo de trabajos relacionados

Autor(es) – Año	Trabajo relacionado	SGBD	Tipo de fragmenta-ción	Dispositivo de pruebas	Lenguaje	*Benchmark*	Esquema de partición	Tipo de operación	Data set
Romero et al., 2015	Estudio acerca de lo viable que es resolver consultas de procesamiento analítico con Hadoop sobre Big Data, beneficiándose de los índices secundarios y el particionamiento con el SGBD HBase.	No especificado	No especificado	No especificado	No especifica-do	No especificado	No especificado	No especificado	No especificado
Qin et al., 2016	Una solución para la carga rápida y sin pérdida de datos, basado en Kafka, HDFS y Spark, por medio de un método de partición basado en la fibra de entidades y un algoritmo de carga en paralelo.	Spark SQL	No especificado	Clúster virtual	SQL	TPC-H	No especificado	Select	No especificado
Kumar, 2016	Estimaciones de rendimiento de las técnicas de recuperación de datos en MySQL, Hive y Apache Pig, mediante un análisis comparativo en función del tiempo de ejecución de las consultas de datos.	MySQL Hive	Horizontal	No especificado	SQL	No especificado	Range List	Insert Select	No especificado
Sun et al., 2016	Técnicas de partición generalizada orientada al salto, desarrollando un novedoso *framework* de omisión de datos híbrido, tomando en cuenta la agrupación de columnas y la selección local de características.	Spark SQL	Horizontal	Clúster virtual	SQL	TPC-H	Range Hash	Select	SDSS DB
Sukhija et al., 2017	Herramienta para el particionamiento automático de los SGBDR utilizando esquemas de partición de Hash, llave o Range haciendo uso de una interfaz interactiva.	MySQL PostgreSQL	Horizontal	PC	Java SQL	No especificado	Range Key Hash	Select Join	USDA DB SDSS DB
Fraczek y Plechawska-Wojcik, 2017	Aplicación Web de una red social que soporta tres tipos de BD: SQL, MongoDB y Apache Cassandra, para un análisis comparativo entre las BD relacionales y NoSQL en el contexto de la lectura y escritura de datos.	PostgreSQL MongoDB Cassandra	No especificado	PC	Java JavaScript SQL CQL	No especificado	Key	Insert Select Update Join	No especificado
Boussahoua et al., 2018	Nuevo método para la construcción de un almacén de datos distribuido, usando una BD NoSQL tipo columna, basado en una estrategia de agrupación de atributos para definir las familias de columnas.	HBase	No especificado	No especificado	No especifica-do	TPC-DS	No especificado	No especificado	No especificado

Autor(es) – Año	Trabajo relacionado	SGBD	Tipo de fragmenta-ción	Dispositivo de pruebas	Lenguaje	*Benchmark*	Esquema de partición	Tipo de operación	Data set
Amirthalingam y Rais, 2018	Nuevo medio para automatizar el particionamiento en Hive, añadiendo un analizador léxico que lee las consultas en Hive para emitir el DDL.	Hive	No especificado	Clúster	SQL	TPC-H	No especificado	Insert Update Delete	No especificado
Suh et al., 2018	Asistente que facilita la partición multinivel, calculando un esquema de partición para una carga de trabajo concreta, por medio de un algoritmo codicioso para la enumeración del espacio de búsqueda sobre los predicados de consulta en la carga de trabajo.	Teradata	Horizontal	PC	Java	SSB	No especificado	Insert Select Delete Update Join	No especificado
Maabreh, 2018	Evaluación de técnicas de partición de datos para mejorar el rendimiento de las consultas en Big Data, aumentando el rendimiento de los SGBD que gestionan bases de datos masivas.	Oracle	Horizontal	No especificado	SQL	No especificado	List Range Hash	Select	Zarqa University DB
Miller et al., 2018	ParDP, herramienta desarrollada en C y SQL, capaz de particionar en paralelo BD relacionales, con el objetivo de mejorar el rendimiento y la escalabilidad.	MySQL PostgreSQL	Horizontal	No especificado	C SQL	No especificado	Range Hash Key	Select Join	The homo_sapiens_core_92_38 DB
Nam et al., 2019	Nuevo método de partición de BD basado en grafos, denominado GPT, para explicar los inconvenientes que generan los esquemas de partición basados en árboles.	Spark SQL	Horizontal	Clúster	SQL	TPC-DS IMDB BioWarehouse	Hash	Select Join	No especificado
Kozma y Morschheuser, 2019	Comparativa entre PostgreSQL y Cassandra centrada en el rendimiento, en el contexto de la latencia y las operaciones registradas por el usuario.	PostgreSQL Cassandra	No especificado	Clúster	SQL	YCSB	No especificado	Insert Select Update Delete	No especificado
Costa et al., 2019	Evaluación del impacto de la partición de datos y el bucketing en sistemas basados en Hive, demostrando las ventajas de implementar Big DWs basados en modelos desnormalizados.	Hive	No especificado	Clúster	SQL	SSB	No especificado	Select Join	No especificado
Khashan et al., 2020	CQNS, *framework* de consultas complejas SQL y NoSQL, que actúa como intérprete enviando las consultas recibidas de cualquier almacén de datos a su motor ejecutable.	Microsoft SQL Server MongoDB Cassandra	No especificado	Clúster	SQL	No especificado	Hash	Insert Select Update Delete Join	No especificado

Autor(es) – Año	Trabajo relacionado	SGBD	Tipo de fragmenta-ción	Dispositivo de pruebas	Lenguaje	*Benchmark*	Esquema de partición	Tipo de operación	Data set
Benkrid et al., 2020	Técnicas inspiradas en la IA para ayudar al diseño físico automatizado de BD, mediante un *framework* basado en un planificador proactivo.	Postgres-XL	No especificado	Clúster	SQL	SSB	Hash	Select Join	No especificado
Mahmud et al., 2020	Estudio exhaustivo de los métodos de particionamiento y muestreo de datos en relación con el procesamiento y el análisis de Big Data.	No especificado	No especificado	No especificado	No especifica-do	No especificado	No especificado	No especificado	No especificado
Sridevi y Sharma, 2020	Estudio acerca de la partición de BD y su finalidad, destacando algunos de los sitios Web y aplicaciones de redes sociales más populares, como Facebook, Twitter, Amazon, WhatsApp e Instagram, analizando algunas de las características clave de los esquemas de partición de cada BD.	No especificado	No especificado	No especificado	No especifica-do	No especificado	No especificado	No especificado	No especificado
Šalgová y Matiaško, 2020	Análisis del particionamiento en relación con las diversas técnicas, métodos y beneficios que aporta, implementando las principales técnicas de partición horizontal: Range, List y Hash.	Oracle	Horizontal	*Laptop*	SQL	No especificado	Range Hash List	Select	No especificado
Šalgová y Matiaško, 2021	El efecto del particionamiento y la indexación en el tiempo de acceso a los datos, comparando escenarios diferentes con distintas combinaciones de particiones creadas sobre tablas e índices.	Oracle	Horizontal	*Laptop*	SQL	No especificado	Range	No especificado	No especificado
Tesis: Caracterización y discriminación de los tipos de fragmentación disponibles en sistemas gestores de bases de datos		MySQL PostgreSQL MongoDB	Horizontal	Computadora de escritorio	SQL Java Json	TPC-H TPC-E	Range List Key Hash	Select Insert Update Delete	No requerido

Capítulo 3. Aplicación de la metodología

3.1. Metodología de investigación

Esta sección presenta la metodología de investigación utilizada para el desarrollo de la tesis la cual se basó en el método científico y se plasmó en cinco etapas: análisis, selección, implementación, comparación y caracterización y discriminación. El seguimiento de cada una de estas etapas permitió llegar a la solución del problema propuesto. A continuación, se presenta en la Figura 3.1 la metodología de investigación utilizada.

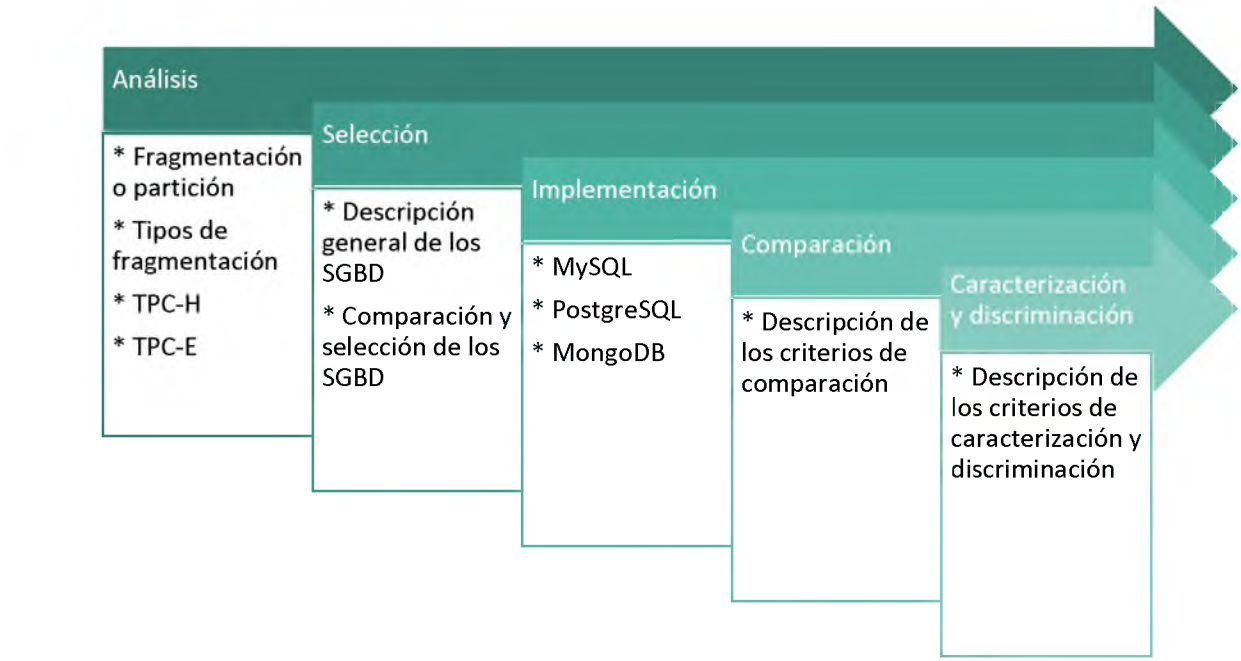


Figura 3.1. Etapas de la metodología de investigación

3.2. Etapa de análisis

3.2.1. Fragmentación o partición

La fragmentación, es una técnica de BD que consiste en dividir una tabla original en tablas más pequeñas (fragmentos) que se consideren necesarios para su uso. Tamer y Valuriez (2020) indican algunas de las razones para usar la fragmentación, entre las que se encuentran:

- Es útil ya que las aplicaciones de BD suelen funcionar con vistas y por ello es posible utilizar distintas relaciones en diferentes nodos para formar la unidad distribuida.
- Gracias a la fragmentación es posible conseguir una mayor eficiencia, dado que los datos suelen estar almacenados cerca del nodo que más utiliza dichos datos.
- Por otra parte, permite aumentar el grado de concurrencia, ya que la fragmentación de las relaciones permite que una transacción se divida en subconsultas que operan sobre estos fragmentos.
- De la misma forma, la fragmentación aporta una mayor seguridad, dado que los datos no utilizados por un nodo local no se almacenan en él, y, por lo tanto, no están al alcance para personas no autorizadas.

Por otro lado, con relación al grado y reglas de fragmentación, Tamer y Valuriez (2020) mencionan que el grado de fragmentación es una decisión importante por tomar antes de que sea aplicada en una BD, ya que afecta el rendimiento al ejecutar las consultas, por lo que una alternativa es no fragmentar nada. Sin embargo, si se decide fragmentar, es importante que se realice de tal manera que sean subconjuntos de tuplas, que es a lo que se le conoce como fragmentación horizontal, o fragmentar a un nivel de subconjuntos de atributos que es a lo que se le conoce como fragmentación vertical.

Además, teniendo en cuenta que la fragmentación en algunos casos afecta el rendimiento de la BD, en especial cuando es usada en distintos fragmentos que están ubicados en diferentes nodos, Tamer y Valuriez (2020) mencionan que es necesario tomar en cuenta los siguientes aspectos:

- **Completitud:** se refiere a una descomposición sin pérdida de datos, es decir, que los datos en una tabla original son mapeados en fragmentos sin pérdida de datos.
- **Reconstrucción:** es la posibilidad de reconstruir una tabla original a partir de fragmentos que aseguran que las restricciones definidas sobre los datos en forma de dependencias se resguarden.

- **Disyunción:** esto establece que, si la tabla original se divide en forma horizontal, ningún dato que se tenga en un fragmento tiene que estar presente en algún otro fragmento de la tabla original.

3.2.2. Tipos de fragmentación

Existen tres tipos de fragmentación: horizontal, vertical e híbrida. Tamer y Valuriez (2020) describen estos tipos de fragmentación como aspectos importantes en el diseño físico de la base de datos, ya que tiene un impacto considerable en el desempeño y la facilidad de administración de BD relacionales.

Fragmentación horizontal

La fragmentación horizontal se realiza sobre las tuplas, concepto definido anteriormente, en donde cada fragmento es un subconjunto de tuplas. Este tipo de fragmentación divide una tabla horizontalmente, agrupando filas para crear un subconjunto de tuplas.

En este contexto, Tamer y Valuriez (2020) mencionan dos variantes de dicha fragmentación: la primaria y la derivada. La fragmentación horizontal primaria, se desarrolla empleando las sentencias definidas de una tabla, mientras que la fragmentación horizontal derivada consiste en dividir una tabla a partir de las sentencias definidas sobre alguna otra tabla.

Fragmentación vertical

Este tipo de fragmentación divide una tabla verticalmente en columnas, en donde cada fragmento mantiene un subconjunto de atributos, así como las claves principales de la tabla. El objetivo principal de dicha fragmentación es dividir una tabla en un conjunto de tablas más pequeñas de tal manera que las aplicaciones son capaces de ejecutarse con un solo fragmento. Por otra parte, dicha fragmentación solo aplica para aquellos atributos que no participan en la llave primaria.

Fragmentación híbrida o mixta

En relación con la fragmentación híbrida o mixta, Tamer y Valuriez (2020) indican que es la unión de las fragmentaciones horizontal y vertical en una sola tabla. En este tipo de fragmentación una tabla se divide en un conjunto de fragmentos de los cuales cada uno contiene su propio subconjunto de atributos y de tuplas de la tabla original. Esto se logra ejecutando primeramente la fragmentación horizontal, seguido de otra fragmentación vertical o viceversa, lo que produce un árbol de particionamiento estructurado.

3.2.3. TPC-H

El *benchmark* TPC-H es una especificación estándar desarrollada por TPC (*Transaction Processing Performance Council*) que sirve como punto de referencia en apoyo a la toma de decisiones. Consiste en un conjunto de consultas *ad hoc* orientadas al negocio y modificaciones de datos concurrentes. Las consultas y los datos que pueblan la base de datos compuesta por ocho tablas fueron elegidos por los creadores del *benchmark* para tener una amplia relevancia en toda la industria. Este punto de referencia sirve de apoyo a los sistemas que examinan grandes volúmenes de datos y que ejecutan consultas con un alto grado de complejidad, además de dar respuestas a preguntas críticas del negocio (Serlin et al., 2021). La Tabla 3.1 muestra las tablas de TPC-H y la cantidad de tuplas para cada una de ellas.

Tabla 3.1. Cantidad de tuplas en cada tabla de la BD TPC-H

Tabla	Cantidad de tuplas
NATION	25
REGION	5
PART	200000
SUPPLIER	10000
PARTSUPP	800000
CUSTOMER	150000
ORDERS	1500000
LINEITEM	6001215

3.2.4. TPC-E

El *benchmark* TPC-E es otro punto de referencia desarrollado por TPC para el procesamiento de transacciones en línea (OLTP, *Online Transaction Processing*). TPC-E es una mezcla de transacciones de lectura y actualizaciones intensivas que simulan las actividades de entornos complejos de aplicaciones OLTP. La base de datos está compuesta por treinta y tres tablas con una amplia gama de columnas, cardinalidad y propiedades de escalamiento. El esquema de la base de datos, la población de datos, las transacciones y las reglas de implementación fueron diseñadas para representar sistemas OLTP modernos (Serlin et al., 2015). La Tabla 3.2 muestra las tablas de TPC-E y la cantidad de tuplas para cada una de ellas.

Tabla 3.2. Cantidad de tuplas en cada tabla de la BD TPC-E

Tabla	Cantidad de tuplas
ACCOUNT_PERMISSION	7128
CUSTOMER	1000
CUSTOMER_ACCOUNT	5000
CUSTOMER_TAXRATE	2000
HOLDING	284290
HOLDING_HISTORY	2247265
HOLDING_SUMMARY	49686
WATCH_ITEM	100602
WATCH_LIST	1000
BROKER	10
CASH_TRANSACTION	1590458
CHARGE	15
COMMISSION_RATE	240
SETTLEMENT	1728000
TRADE	1728000
TRADE_HISTORY	4148112
TRADE_REQUEST	0
TRADE_TYPE	5
COMPANY	500
COMPANY_COMPETITOR	1500
DAILY_MARKET	893925
EXCHANGE	4
FINANCIAL	10000
INDUSTRY	102
LAST_TRADE	685

Tabla	Cantidad de tuplas
NEWS_ITEM	1000
NEWS_XREF	1000
SECTOR	12
SECURITY	685
ADDRESS	1504
STATUS_TYPE	5
TAXRATE	320
ZIP_CODE	14741

3.3. Etapa de selección

3.3.1. Descripción general de los SGBD

Oracle

Oracle Database es un SGBD relacional de uso comercial desarrollado en C y C++. Es la BD más popular del mundo de acuerdo con (*DB-Engines*, 2021) y es desarrollado, distribuido y soportado por Oracle.

En Oracle Database las licencias son comerciales, pero actualmente cuenta con una versión gratuita que no incluye algunas plataformas de desarrollo como Oracle *Developer Tools* para Visual Studio y el desarrollador Oracle SQL. Oracle ofrece distintas características y precios de acuerdo con la versión que se desee adquirir. Entre sus ediciones más populares se encuentran:

- **Oracle Database Express Edition:** está disponible en plataformas Linux x86-64 y Microsoft Windows, y se publica bajo los términos y condiciones de uso gratuito de Oracle.
- **Oracle Database Standard Edition 2:** incluye las características necesarias para desarrollar aplicaciones Web, de grupo de trabajo, de departamento y de grupo.
- **Oracle Database Enterprise Edition:** proporciona rendimiento, disponibilidad, escalabilidad y seguridad para el desarrollo de aplicaciones como aplicaciones de procesamiento de transacciones en línea de gran volumen, almacenes de datos con uso intensivo de consultas y aplicaciones de Internet exigentes.

- **Oracle Database Personal Edition:** admite entornos de desarrollo e implementación de un solo usuario que requieren compatibilidad total con Oracle Database Standard Edition 2 y Oracle Database Enterprise Edition.

Adicionalmente, Oracle Database implementa características orientadas a objetos, como tipos definidos por el usuario, herencia y polimorfismo, por lo que se denomina como un SGBD relacional de objetos. Así mismo, extendió el modelo relacional a un modelo objeto-relacional, lo que permite almacenar modelos de negocio complejos en una BD relacional. Oracle unifica las tareas básicas de SQL dando origen a Oracle SQL, que es una implementación del estándar ANSI (*American National Standards Institute*) y admite numerosas características que van más allá del SQL estándar.

Por otra parte, está dotado de una seguridad de BDs avanzada, lo que reduce el riesgo de violaciones de datos. Del mismo modo, cumple con los requisitos de rendimiento en entornos de tiempo real e implementaciones de centros de datos, lo que le brinda una gama de capacidades diseñadas para optimizar la baja latencia y el alto rendimiento.

Así mismo, los desarrolladores tienen la posibilidad de crear rápidamente aplicaciones escalables y de alto rendimiento utilizando SQL, JSON, XML y una variedad de lenguajes de programación, como C, C++, Java, Python, entre otros. Oracle Database 21c ofrece una gama de herramientas de desarrollo integradas, como APEX (*Oracle Application Express*), y capacidades de bases de datos convergentes.

En Oracle Database, cada BD (CDB por sus siglas en inglés *Container Database*) contiene o es contenida por otra BD (PDB por sus siglas en inglés *Pluggable Database*). Oracle Sharding es una técnica de escalamiento de BDs basada en la partición horizontal de datos en múltiples PDB como una única BD lógica. En esta arquitectura de fragmentación, cada CDB se aloja en un servidor dedicado con sus propios recursos locales: CPU, memoria, unidad *flash* o disco. Adicionalmente, es posible designar una PDB como fragmento. Los fragmentos de PDB de diferentes CDBs forman una única BD lógica, denominada BD fragmentada.

Dos fragmentos del mismo CDB no necesariamente son miembros de la misma BD fragmentada. Sin embargo, dentro del mismo CDB, es posible que una PDB se encuentre en una BD fragmentada, y otra PDB se encuentre en otra BD fragmentada.

Además, Oracle admite una amplia gama de métodos de partición como: Range, List y Hash, al igual que MySQL, pero además cuenta con otros métodos de partición como:

- ***Auto-List Partitioning:*** amplía las capacidades del método List al definir automáticamente nuevas particiones para cualquier nuevo valor de clave de partición.
- ***Composite Partitioning:*** se utilizan combinaciones de dos métodos de distribución de datos. Primero, la tabla se particiona mediante el método de distribución de datos uno y luego cada partición se subdivide en subparticiones utilizando el segundo método de distribución de datos.
- ***Multi-Column Range Partitioning:*** una opción cuando la clave de partición se compone de varias columnas y las columnas subsiguientes definen un nivel de granularidad mayor que las anteriores.
- ***Interval Partitioning:*** amplía las capacidades del método de Range al definir automáticamente rangos particionados equitativamente para cualquier partición futura utilizando una definición de intervalo como parte de los metadatos de la tabla.
- ***Reference Partitioning Partitions:*** una tabla que aprovecha una relación padre-hijo existente. La relación de clave principal se utiliza para heredar la estrategia de partición de la tabla principal a su tabla secundaria.
- ***Virtual Column Based Partitioning:*** permite que la clave de partición sea una expresión, utilizando una o más columnas existentes de una tabla y almacenando la expresión solo como metadatos.
- ***Interval Reference Partitioning:*** una extensión del particionamiento de referencia que permite el uso de tablas particionadas de intervalo como tablas principales para la partición de referencia (*Oracle*, 2021).

MySQL

MySQL es un SGBD relacional de código abierto (*Open Source*) escrito en C y C++. Ocupa el segundo lugar en popularidad de acuerdo con (*DB-Engines*, 2021) y es desarrollado, distribuido y soportado por *Oracle Corporation*. MySQL es una marca comercial de *Oracle Corporation* y/o sus filiales.

Así mismo, MySQL cuenta con doble licencia. Los usuarios tienen la oportunidad de optar por utilizar el software MySQL como un producto de código abierto bajo los términos de la Licencia Pública General (GPL por sus siglas en inglés *General Public License*) GNU, o comprar una licencia comercial estándar de Oracle. MySQL es de código abierto, por lo que cualquier usuario descarga el software de Internet y lo utiliza sin pagar nada. Del mismo modo, es posible estudiar el código fuente y cambiarlo para que se adapte a las necesidades del usuario.

Adicionalmente, MySQL utiliza el lenguaje estandarizado SQL para acceder a las BDs. Dependiendo del entorno en que se esté programando es posible escribir SQL directamente, incrustar instrucciones SQL con código escrito en otro lenguaje o usar una API específica del lenguaje que oculte la sintaxis SQL.

Por otra parte, si lo que se busca es rapidez, confiabilidad, escalabilidad y un fácil manejo del software, MySQL es una buena opción. MySQL Server se ejecuta cómodamente en una computadora de escritorio o portátil, junto con otras aplicaciones, servidores Web, entre otros, que además requiere de poca o nula atención. Si se dedica una máquina completa a MySQL, es posible ajustar la configuración para aprovechar toda la memoria, la potencia de la CPU y la capacidad de E/S disponibles. MySQL también es escalable a clústeres de máquinas en red.

Originalmente, MySQL Server se desarrolló para manejar grandes BDs de forma más rápida que las soluciones existentes teniendo éxito en entornos de producción altamente exigentes durante varios años. Aunque sigue en constante desarrollo, hoy en día MySQL ofrece un conjunto rico y útil de funciones. Su conectividad, velocidad y seguridad, hacen que MySQL Server sea muy adecuado para acceder a BDs en Internet.

Además, MySQL Server tiene un conjunto práctico de características desarrolladas en estrecha colaboración con los usuarios, por lo que tiene gran compatibilidad con aplicaciones y lenguajes. Las APIs para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby y Tcl están disponibles, lo que permite que los clientes MySQL se escriban en muchos lenguajes.

Por otra parte, en relación con el particionamiento, MySQL 8.0 no admite actualmente la partición de tablas de forma nativa, por lo que el soporte de particionamiento es proporcionado por los motores de almacenamiento InnoDB y NDB. MySQL solo admite particionamiento horizontal, para lo cual dispone de los siguientes tipos:

- ***Range partitioning*:** Este tipo de particionamiento asigna filas a las particiones en función de los valores de columna que se encuentran dentro de un rango determinado.
- ***List partitioning*:** Similar al *Range partitioning*, excepto que la partición se selecciona en función de columnas que coincidan con una de un conjunto de valores discretos.
- ***Hash partitioning*:** En este tipo de particionamiento, se selecciona una partición en función del valor devuelto por una expresión definida por el usuario que opera sobre los valores de columna de las filas que se insertan en la tabla.
- ***Key partitioning*:** Este tipo de partición es similar al *Hash partitioning*, excepto que solo se proporcionan una o más columnas a evaluar y el servidor MySQL proporciona su propia partición Hash (*MySQL*[2], 2021).

Microsoft SQL Server

Microsoft SQL Server es un SGBD relacional de uso comercial escrito en C++. Ocupa el tercer lugar en popularidad de acuerdo con (*DB-Engines*, 2021) y es desarrollado, distribuido y soportado por Microsoft.

En cuanto a las licencias de uso, SQL Server se ofrece en dos ediciones principales para adaptarse a los requisitos únicos de precios, rendimiento y características de organizaciones y usuarios:

- ***Enterprise Edition***: es ideal para aplicaciones que necesitan rendimiento en memoria, seguridad y alta disponibilidad.
- ***Standard Edition***: ofrece funciones de base de datos completas para aplicaciones y *data marts* (almacén de datos orientado a un área específica) de nivel medio.

Así mismo, cuenta con ediciones gratuitas como *SQL Server Developer edition*, que permite a los desarrolladores crear cualquier tipo de aplicación sobre SQL Server con toda la funcionalidad de la edición Enterprise, pero su licencia brinda uso solo como sistema de desarrollo y prueba. Así también, cuenta con la *Express edition*, que es la BD gratuita de nivel básico y es ideal para el aprendizaje y la creación de aplicaciones de escritorio y de pequeños servidores de datos.

SQL Server es uno de los SGBDs relacionales más instalados del mundo. El lenguaje de desarrollo utilizado es Transact-SQL, que es una implementación del estándar ANSI del lenguaje SQL, utilizado para manipular y recuperar datos, crear tablas y definir relaciones entre ellas.

Como bien se sabe, la seguridad es esencial en cualquier SGBD y SQL Server es la BD más segura en los últimos ocho años, según el Instituto Nacional de Estándares y Tecnología. Del mismo modo, SQL Server 2019 añade compatibilidad con los enclaves seguros siempre encriptados, para permitir cálculos ricos en datos encriptados.

En cuanto a la compatibilidad con lenguajes y plataformas, SQL Server es compatible con lenguajes de programación como T-SQL, Java, C/C++, Scala, Node.js, C#/VB.NET, Python, Ruby y .NET Core. Así mismo, SQL Server admite el aprendizaje automático y la extensibilidad con R, Python, Java y Microsoft.net.

Por otra parte, a partir del lanzamiento de SQL Server 2005, SQL Server incluye el particionamiento de tablas e índices, lo que permite que los datos de las tablas e índices particionados se repartan entre varios grupos de archivos de una BD. No obstante, solo se admite la partición por columna (*Column Partitioning o Range Partitioning*), que consiste en tomar una columna de una tabla o índice para crear particiones en los mismos (*Microsoft*, 2021).

PostgreSQL

PostgreSQL es un SGBD relacional de código abierto escrito en C y es desarrollado, distribuido y soportado por *PostgreSQL Global Development Group*. Ocupa el cuarto lugar en popularidad de acuerdo con (*DB-Engines*, 2021) y está basado en POSTGRES, versión 4.2. POSTGRES fue pionero en muchos conceptos que estuvieron disponibles mucho más tarde en algunos sistemas de BDs comerciales.

Gracias a la licencia libre, al igual que MySQL, PostgreSQL también permite ser utilizado, modificado y distribuido por cualquier persona de forma gratuita y para cualquier propósito, ya sea privado, comercial o académico.

Así mismo, PostgreSQL es un SGBD relacional, por lo que es compatible con una gran parte del estándar SQL y ofrece muchas características modernas, como consultas complejas y vistas actualizables. Del mismo modo, admite los tipos SQL estándar, así como otros tipos de utilidad general y un rico conjunto de tipos geométricos.

Además, PostgreSQL cuenta con la posibilidad de ser ampliado por el usuario de muchas maneras, por ejemplo, adicionando nuevos tipos de datos, funciones, operadores, entre otros.

En cuanto a la compatibilidad con lenguajes y aplicaciones, libpq es el motor subyacente para la conexión con varias interfaces de aplicaciones, que incluye las escritas para C++, Perl, Python, Tcl y ECPG. Es una biblioteca con un conjunto de funciones que permiten a los programas cliente pasar consultas al servidor *back-end* de PostgreSQL y recibir los resultados de estas consultas.

Por otro lado, PostgreSQL ofrece soporte integrado para el particionamiento de tablas con Range, List y Hash, aunque no es posible convertir una tabla normal en una tabla particionada o viceversa. Sin embargo, es posible agregar una tabla regular o particionada existente como una partición de una tabla particionada, o eliminar una partición de una tabla particionada convirtiéndola en una tabla independiente, lo que simplifica y acelera muchos procesos de mantenimiento (*PostgreSQL*[2], 2021).

MongoDB

MongoDB es un SGBD no relacional de código abierto orientado a documentos. Es desarrollado en C++ y ocupa el quinto lugar en popularidad de acuerdo con (*DB-Engines*, 2021).

Las licencias de MongoDB son de código abierto y se obtienen de forma gratuita bajo la licencia pública general de Affero (AGPL) GNU. Cuenta con dos versiones principales: MongoDB Community Server, que ofrece soporte para consultas *ad hoc*, indexación secundaria y agregaciones en tiempo real que proporciona formas poderosas para acceder y analizar los datos y MongoDB Enterprise Server, que es la edición comercial que incluye capacidades adicionales como el motor de almacenamiento en memoria para un alto rendimiento y baja latencia, características avanzadas como controles de acceso LDAP (protocolo ligero de acceso a directorios) y Kerberos, así como de cifrado para datos en reposo.

Además, MongoDB es un SGBD NoSQL. Por su parte, MongoDB almacena registros de datos como documentos BSON, similar a los objetos JSON, que se reúnen en colecciones. Una BD en MongoDB, almacena una o más de estas colecciones.

Por otra parte, MongoDB proporciona compatibilidad con modelos de datos integrados, lo que reduce la actividad de E/S en el sistema. Además, los índices admiten consultas más rápidas e incluyen claves de documentos y matrices incrustados en las consultas.

MongoDB es compatible con múltiples aplicaciones y lenguajes, como C, C++, Haskell, Java, JavaScript, Python, MATLAB, entre otros. Así mismo, está presente en entornos virtuales como: VMware y KVM (*Kernel-based Virtual Machine*).

Además, MongoDB proporciona escalabilidad horizontal como parte de su funcionalidad principal. Esto lo logra distribuyendo los datos a distintos nodos de un clúster. Así mismo, admite la creación de zonas de datos basadas en la clave de partición y gracias a esto, MongoDB es capaz de dirigir las lecturas y escrituras cubiertas por una zona solo a los fragmentos dentro de la zona (*MongoDB*[1], 2021).

Redis

Redis es un SGBD no relacional de código abierto. Ocupa el sexto lugar en popularidad de acuerdo con (*DB-Engines*, 2021). Está implementado en lenguaje C por Salvatore Sanfilippo, quien es patrocinado por Redis Ltd.

Por su parte, Redis es un software de código abierto lanzado bajo los términos de la licencia BSD (*Berkeley Software Distribution*) de tres cláusulas. La mayor parte del código fuente de Redis fue escrito y tiene derechos de autor de Salvatore Sanfilippo y Pieter Noordhuis.

En su interior, Redis es un almacén de estructura de datos en memoria utilizado principalmente como BD, basado en el almacenamiento de tablas por clave/valor. No obstante, también es utilizado como caché, agente de mensajes y motor de *streaming*. Redis proporciona estructuras de datos como cadenas, hashes, listas, conjuntos, conjuntos ordenados con consultas de rango, mapas de bits, entre otros. Así mismo cuenta con replicación incorporada, *scripts* Lua, desalojo de LRU (algoritmo *Least Recently Used* que descarta los elementos recientes menos usados), transacciones y diferentes niveles de persistencia en disco, y proporciona alta disponibilidad a través de Redis Sentinel y particionamiento automático con Redis Cluster.

La granularidad de partición es la clave, por lo que no es posible fragmentar un conjunto de datos con una sola clave enorme como un conjunto ordenado muy grande. Tiene la ventaja de escalar la potencia computacional a múltiples núcleos y múltiples computadoras, y el ancho de banda de red a múltiples computadoras y adaptadores de red.

Así mismo, Redis funciona en la mayoría de los sistemas POSIX (norma escrita por la IEEE, que define una interfaz estándar del sistema operativo y el entorno, incluyendo un intérprete de comandos) como Linux, *BSD y OS X, sin dependencias externas. Linux y OS X son los dos sistemas operativos donde más se desarrolla y prueba Redis, por lo que se recomienda usar Linux para la implementación. Redis funciona en sistemas derivados de Solaris como SmartOS, pero el soporte es el mejor esfuerzo. No hay soporte oficial para las compilaciones

de Windows. Los lenguajes en los que se implementa son: C#, C, C++, Clojure, Crystal, D, Dart, Elixir, Fancy, entre otros.

Además, la partición en Redis permite BD mucho más grandes, utilizando la suma de la memoria de muchas computadoras. Sin particionar, está limitado a la cantidad de memoria que proporciona una sola computadora. Redis Cluster es la forma preferida de obtener fragmentación automática y alta disponibilidad, es una mezcla entre enrutamiento de consultas y particionamiento del lado del cliente, pero además cuenta con particionamiento Hash. No obstante, algunas características de Redis no funcionan muy bien con el particionamiento (*Redis*, 2021).

IBM Db2

IBM Db2 es una familia de productos de gestión de datos, que incluye el SGBD relacional de uso comercial Db2. Es desarrollado, distribuido y soportado por IBM y fue escrito en C y C++. Además, ocupa el séptimo lugar en popularidad de acuerdo con (*DB-Engines*, 2021).

Además, Db2 cuenta con una serie de ediciones de productos diseñadas para escalar a las actuales necesidades empresariales. Aunque es comercial, también cuenta con una versión gratuita. Algunas de sus ediciones principales son:

- **Db2 Community Edition:** es una edición sin costo alguno, que cuenta con las características de nivel de entrada del servidor de datos Db2, útil para la comunidad de desarrolladores y socios.
- **Db2 Advanced Edition:** proporciona una solución de base de datos integral para la empresa con una métrica de licencias para facilitar la implementación híbrida en la nube. Además, está disponible con una licencia de software perpetua y una suscripción mensual para uso de producción y no producción sin restricciones y con soporte *premium* de IBM.

Por otra parte, Db2 es un SGBD relacional que utiliza el lenguaje estándar SQL. Sin embargo, es compatible con SQL PL, que es una extensión del lenguaje SQL y contiene instrucciones y elementos de lenguaje que se usan para implementar la lógica de procedimiento en instrucciones SQL.

La mayor parte de la familia Db2 está disponible en la plataforma IBM Cloud Pak for Data, ya sea como complemento o como servicio de orígenes de datos incluido, de modo que prácticamente todos los datos están disponibles en los entornos híbridos o *multicloud* para las aplicaciones de IA. Cuando la IA se integra en un SGBD, se resuelven algunos desafíos de datos al mejorar la precisión y el rendimiento de las consultas de la BD y optimizar los recursos del sistema.

Al igual que la mayoría de los SGBDs, Db2 es compatible con múltiples aplicaciones y lenguajes, como Java, JSON, Node.js, Perl, PHP, Python, Scala, Ruby.

Por su parte, Db2 maneja configuraciones de partición de BD única y de particiones de BD múltiples. Los primeros incluyen configuraciones de un solo procesador y de múltiples procesadores, los segundos incluyen particiones de BD con uno o múltiples procesadores y particiones de BD lógicas. Además, Db2 admite el particionamiento Range que es un tipo de particionamiento horizontal (*IBM*, 2021).

Elasticsearch

Elasticsearch es un SGBD no relacional de código abierto basado en Apache Lucene y escrito en Java. Ocupa el octavo lugar en popularidad de acuerdo con (*DB-Engines*, 2021) y es desarrollado, distribuido y soportado por la empresa Elastic.

Por otra parte, Elasticsearch está publicado como código abierto bajo las condiciones de la licencia de Apache. Así mismo, Elasticsearch es una BD NoSQL, que además es muy popular para el manejo de Big Data.

Por otro lado, al estar desarrollado en Java, es compatible en todas las plataformas donde Java lo sea. Además, utiliza objetos JSON como respuesta, por lo que es fácil de invocar desde varios lenguajes de programación, aunque esto quizás resulta en una desventaja, ya que solo soporta este tipo de respuesta, lo que lo limita al uso de algunos lenguajes como XML.

En cuanto a la compatibilidad con aplicaciones y lenguajes, Elasticsearch es compatible con .Net, Groovy, Java, JavaScript, Perl, PHP, Python y Ruby, además

de que se ejecuta en prácticamente todos los sistemas operativos con Java VM (máquina virtual de Java).

Además, Elasticsearch soporta el particionamiento *Sharding*, utilizado para almacenar diferentes datos en distintos nodos. Por lo que, en la actual versión 7.5, Elasticsearch comprime el tráfico de red y optimiza la recuperación de *shards* para reducir los costos de Elastic Cloud y mejorar la resistencia de los datos (*Elastic*, 2021).

SQLite

SQLite es un SGBD relacional de dominio público escrito en C. Ocupa el noveno lugar en popularidad de acuerdo con (*DB-Engines*, 2021). SQLite es la BD más implementada en el mundo con más aplicaciones de las que podemos contar, incluidos varios proyectos de alto perfil.

Adicionalmente, SQLite es un motor de BDs con SQL integrado. A diferencia de la mayoría de las otras bases de datos SQL, SQLite no tiene un proceso de servidor independiente. SQLite lee y escribe directamente en archivos de disco ordinarios. Una base de datos SQL completa con varias tablas, índices, desencadenadores y vistas, está contenida en un solo archivo de disco. SQLite generalmente se ejecuta más rápido en cuanto más memoria se le proporcione.

Por otra parte, SQLite se prueba muy cuidadosamente antes de cada lanzamiento y tiene la reputación de ser muy confiable. Del mismo modo, la mayor parte del código fuente se dedica exclusivamente a pruebas y verificación lo que le brinda un conjunto de pruebas automatizadas que ejecutan millones de casos de prueba.

SQLite tiene un conjunto práctico de características desarrolladas en estrecha colaboración con un equipo internacional de desarrolladores que trabajan en SQLite a tiempo completo por lo que tiene gran compatibilidad con aplicaciones y lenguajes, como SQL, C, C#, C++, TLC y R. En cuanto al particionamiento, SQLite no admite actualmente particionamiento de ningún tipo (*SQLite*, 2021).

Cassandra

Cassandra es un SGBD no relacional de código abierto escrito en Java. Ocupa el décimo lugar en popularidad de acuerdo con (*DB-Engines*, 2021) y está soportada por Apache Software Fundation.

Apache Cassandra se diseñó inicialmente en Facebook utilizando una arquitectura basada en eventos por etapas (SEDA) para implementar una combinación de las técnicas de almacenamiento distribuido y replicación Dynamo de Amazon y el modelo de motor de datos y almacenamiento Bigtable de Google. Cassandra se diseñó como la mejor combinación de ambos sistemas para satisfacer los nuevos requisitos de almacenamiento a gran escala, tanto en lo que respecta a la huella de datos como al volumen de consultas. A medida que las aplicaciones comenzaron a requerir una replicación global completa y lecturas y escrituras de baja latencia siempre disponibles, se hizo imperativo diseñar un nuevo tipo de modelo de BD, ya que los SGBDs relacionales de la época se esforzaban por satisfacer los nuevos requisitos de las aplicaciones de escala global.

Las BDs NoSQL permiten una organización y análisis rápidos y *ad hoc* de tipos de datos dispares y de volumen extremadamente alto. Este factor es más importante en los últimos años, con el advenimiento de Big Data y la necesidad de escalar rápidamente las BDs en la nube. Por diseño, las bases de datos NoSQL son ligeras, de código abierto, no relacional y distribuido en gran medida. Entre sus puntos fuertes se cuentan la escalabilidad horizontal, las arquitecturas distribuidas y un enfoque flexible para la definición de esquemas.

Un atributo importante de Cassandra es que sus bases de datos están distribuidas. Eso produce ventajas técnicas y comerciales. Las bases de datos Cassandra se escalan fácilmente cuando una aplicación está bajo alto estrés, y la distribución también evita la pérdida de datos por la falla de hardware de cualquier centro de datos dado. Una arquitectura distribuida también aporta poder técnico, un desarrollador tiene permitido modificar el rendimiento de las consultas de lectura o escribir consultas de forma aislada. Cassandra permite a los desarrolladores escalar sus bases de datos dinámicamente, utilizando hardware estándar, sin tiempo de

inactividad. Debido a que se basa en nodos, Cassandra escala horizontalmente, utilizando hardware básico inferior.

En Cassandra, los datos en sí se distribuyen automáticamente, con consecuencias de rendimiento positivas. Esto lo logra utilizando particiones, además cada nodo posee un conjunto particular de *tokens*, y Cassandra distribuye datos basados en los rangos de estos *tokens* en todo el clúster. La clave de partición es responsable de distribuir los datos entre los nodos y es importante para determinar la localidad de los datos. Cuando se insertan datos en el clúster, el primer paso es aplicar una función Hash a la clave de partición. La salida se utiliza para determinar qué nodo (basado en el rango de *tokens*) obtiene los datos (*Cassandra*, 2021).

3.3.2. Comparación y selección de los SGBD

Una vez presentados y descritos brevemente los distintos SGBDs, en la Tabla 3.3 se presenta un análisis comparativo, en donde se describen diversos aspectos que fueron considerados para identificar las similitudes y particularmente las principales características que ofrecen en cuanto al particionamiento de tablas, que hacen de ese SGBD un candidato de estudio para implementar los tipos de partición en las bases de datos TPC-H y TPC-E.

- SGBD. - se coloca el nombre del sistema gestor de base de datos.
- Versión. - se indica la versión actual del software del SGBD.
- Modelo de datos. - se indica el modelo de datos (relacional, no relacional).
- Tipo de licencia. - se indica el tipo de licencia de uso que maneja el SGBD.
- Tipo de fragmentación. – se indican los tipos de fragmentación que soporta el SGBD.
- Lenguaje. – se indican el lenguaje de consulta utilizado para el acceso a los datos por el SGBD.
- Esquema de partición. – se indican los esquemas de partición que soporta el SGBD.
- Compatibilidad con lenguajes. – se indican los lenguajes con los cuales es compatible el SGBD.

- Compatibilidad con plataformas. – se indican las plataformas con las que es posible instalar el SGBD.

El resultado del análisis comparativo indica que el 90% de los SGBDs es compatible con plataformas Linux, Unix y Windows. Así mismo, el 60% de los SGBDs maneja el modelo de datos relacional (SQL) y solo un 40% el modelo no relacional (NoSQL). Por otra parte, se identificó que el 70% de los SGBDs es de código abierto, y aunque un 40% es de uso comercial, cuentan con una versión libre. Además, se encontró también que el 90% soporta al menos el particionamiento horizontal, aunque existe ausencia del particionamiento vertical e híbrido.

En cuanto al lenguaje de consulta utilizado por los SGBDs, un 80% utiliza el lenguaje SQL para el acceso a los datos y en menor medida, se utilizan algunas variantes de SQL, como PL/SQL (lenguaje de programación estructurado de Oracle), CQL (*Cassandra Query Language*) y EQL (lenguaje de consulta para series temporales basadas en eventos de ElasticSearch). Así mismo, en cuanto a los esquemas de particionamiento que manejan los SGBDs, se encontró que el particionamiento Range es admitido por el 60% de estos gestores y en menor medida, el particionamiento Hash y List. No obstante, SGBDs como MySQL y Oracle, admiten más esquemas de partición, como el particionamiento Key, por ejemplo. Además, todos los SGBDs tienen compatibilidad con una gran variedad de lenguajes de programación, siendo C, C++ Java y Python los que más compatibilidad tienen con estos, y en menor medida, PHP, Ruby, C#, entre otros.

Gracias a este estudio, se lograron identificar y seleccionar tres SGBDs en donde se implementaron los tipos de partición en la base de datos TPC-H y TPC-E. Primeramente, se seleccionó al SGBD MySQL, el cual a pesar de ser de código abierto es un muy buen candidato, ya que, aunque no cuenta con la misma cantidad de esquemas de partición como Oracle, si ofrece más opciones que el resto de los SGBDs. Además, también se seleccionó PostgreSQL, que también es un buen SGBD relacional. Dentro de la categoría de SGBD NoSQL, el SGBD seleccionado fue MongoDB, ya que ofrece más opciones de particionamiento en comparación con los otros SGBD NoSQL.

Tabla 3.3. Comparación de los SGBDs

SGBD	Versión	Modelo de datos	Tipo de licencia	Tipo de fragmentación	Lenguaje	Esquema de partición	Compatibilidad con lenguajes	Compatibilidad con plataformas
Oracle	21c	Relacional	Comercial	Horizontal	SQL PL/SQL Java	Range List Auto-list Hash Composite Multi-column Range Interval Reference Virtual column Interval reference	C C++ C# COBOL Fortran Java JavaScript PHP Python Ruby R	Linux Unix Windows
MySQL	8.0	Relacional	Código abierto Comercial	Horizontal	SQL	Range List Hash Key	C C++ Eiffel Java Perl PHP Python Ruby Tcl	Linux Unix Windows
Microsoft SQL Server	2019	Relacional	Comercial	Horizontal	SQL	Range	Transact-SQL Java C C++ C# Scala JavaScript Visual Basic Python R Ruby .NET Core	Windows Linux Docker
PostgreSQL	14.1	Relacional	Código abierto	Horizontal	SQL	Range List Hash	C C++ Java JavaScript Perl Python Tcl	Linux Unix Windows

SGBD	Versión	Modelo de datos	Tipo de licencia	Tipo de fragmentación	Lenguaje	Esquema de partición	Compatibilidad con lenguajes	Compatibilidad con plataformas
MongoDB	5.0	No relacional	Código abierto	Horizontal	SQL – Solo lectura BSON	Range Hash	C C++ C# .NET Java JavaScript Node.js Perl PHP Python Ruby Scala Delphi	Linux Unix Windows
Redis	6.2.6	No relacional	Código abierto	Horizontal	Lua	Hash Redis Cluster	C C++ C# Clojure Erlang Go Haskell Java Objective-C Perl PHP Python Ruby Scala Smalltalk Tcl.	Linux Unix Windows Docker
IBM Db2	11.5.6	Relacional	Comercial	Horizontal	SQL PL/SQL	Range	C C# C++ COBOL Delphi Fortran Java Perl PHP Python Ruby Visual Basic	Linux Unix Windows

SGBD	Versión	Modelo de datos	Tipo de licencia	Tipo de fragmentación	Lenguaje	Esquema de partición	Compatibilidad con lenguajes	Compatibilidad con plataformas
Elasticsearch	7.15	No relacional	Código abierto	Horizontal	SQL EQL	Sharding	Curl C# Go Java JavaScript Perl PHP Python Ruby	Linux Unix Windows
SQLite	3.37.0	Relacional	Código abierto	No aplica	SQL	No aplica	Basic C C# C++ Tcl R	Windows Linux Unix Android iOS
Cassandra	4.0	No relacional	Código abierto	Horizontal	CQL Lua	Hash	Java Python Ruby C# / .NET Node.js PHP C++ Scala Clojure Erlang Go Haskell Rust Perl Elixir Dart	Linux Unix Windows Docker

3.4. Etapa de implementación

Para la etapa de implementación, es importante destacar que, para el desarrollo y experimentos elaborados en esta tesis, solo se utilizó la base de datos proporcionada por cada *benchmark* (TPC-H y TPC-E), no se llevó a cabo la ejecución del *benchmark* tal como se describe en la documentación de cada uno de ellos, por el contrario, se siguieron e implementaron técnicas y metodología propias para medir el rendimiento de cada base de datos acorde a los tipos de fragmentación que cada SGBD proporciona. Además, en el caso de TPC-H se utilizaron las 22 consultas que el *benchmark* proporciona y para TPC-E se utilizaron ocho consultas propuestas para esta BD. Los archivos y códigos utilizados para la generación de cada una de las bases de datos se presentan en el anexo A.

3.4.1. MySQL

3.4.1.1. Instalación y configuraciones previas

Primeramente, se descargó MySQL Community en su versión 8.0 directamente de su página oficial (MySQL Downloads, 2022). Esta versión es la más actual a la fecha en que se realiza esta implementación y es la versión gratuita bajo los términos de licencia GPL, además el paquete se encuentra dotado de diversas herramientas además del servidor de MySQL; como el Workbench, que es una herramienta visual para la conexión con MySQL.

Posteriormente, se instaló MySQL en el equipo donde se realizaron las pruebas y se realizaron las siguientes configuraciones previas para poder generar la BD TPC-H y TPC-E de forma correcta.

- Se editó el archivo de configuración de MySQL, que por lo general se encuentra ubicado en “C:\ProgramData\MySQL\MySQL Server 8.0”. En este archivo se modificó el atributo “secure-file-priv” asignándole comillas dobles (“”). Esta variable se modificó, ya que por defecto MySQL solo permite cargar tablas desde los archivos que se encuentren en la ruta especificada en ese atributo y al poner comillas

dobles, se indica al servidor que permita cargar los archivos desde cualquier ubicación.

- Posteriormente, se reinició el servicio MySQL80, que es el proceso encargado de manejar el servidor de MySQL en Windows, con la utilidad services.msc.
- Además, se agregó a las variables de entorno de Windows, específicamente a la variable *path*, la ruta donde se encuentran los archivos ejecutables de MySQL, que por lo general se encuentra en "C:\Program Files\MySQL\MySQL Server 8.0\bin" y así poder ejecutar los comandos MySQL de forma rápida desde cualquier terminal.

3.4.1.2. Generación y llenado de la BD TPC-H original

Se generó en primera instancia, el esquema que contendrá la BD original haciendo uso del comando "create database". Posteriormente, se generaron cuatro bases de datos más, para almacenar las BD en donde se implementó cada tipo de particionamiento o fragmentación. Para ello, se utilizará a lo largo de la etapa de implementación, los siguientes nombres para referirse a las BD TPC-H.

- **tpch**: base de datos original de TPC-H
- **tpch_range**: base de datos para el particionamiento Range
- **tpch_list**: base de datos para el particionamiento List
- **tpch_hash**: base de datos para el particionamiento Hash
- **tpch_key**: base de datos para el particionamiento Key

Teniendo los esquemas de BD creados, se pasó a crear las tablas en la BD original. 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 ninguna modificación en él para el caso de MySQL como se muestra en la Figura 3.2. El *script* se ejecutó haciendo uso del comando "source" o "\." De MySQL, que permite ejecutar *scripts* SQL de la ruta proporcionada, tal como se describe en la Figura 3.3.

```
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 );
```

Figura 3.2. Fragmento del *script* que crea las tablas de la BD TPC-H en 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)
```

Figura 3.3. Creación de tablas en la BD TPC-H en MySQL

Teniendo las tablas creadas, se agregaron las relaciones que también las proporciona el *benchmark*, al cual tampoco se le hizo modificación alguna en el caso de MySQL, tal como se observa en la Figura 3.4. Además, en la Figura 3.5 se observa la ejecución del *script* que crea las relaciones.

```
-- 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);


-- For table PART

ALTER TABLE PART
ADD PRIMARY KEY (P_PARTKEY);
```

Figura 3.4. Fragmento del *script* que crea las relaciones de la BD TPC-H en MySQL

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

Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Figura 3.5. Creación de relaciones en la BD TPC-H en MySQL

Una vez generada la BD, junto con las tablas y las relaciones, se procedió a realizar el llenado de la BD haciendo uso del comando "LOAD DATA INFILE" de MySQL. En este caso, se desarrolló un pequeño *script* como se muestra en la Figura 3.6, que automatiza el proceso de llenado de las tablas, ya que el *benchmark* solo proporciona los archivos en texto plano que contienen las tuplas para cada tabla de la BD TPC-H. Además, los atributos se encuentran separados por el símbolo *pipe*

"|" en los archivos que proporciona el *benchmark*. Además, la Figura 3.7 muestra la ejecución en consola de dicho *script*.

```
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;
```

Figura 3.6. *Script* que carga las tuplas en la BD TPC-H en MySQL

```
mysql> source D:\Documentos\Suriel\Tesis\MySQL\cargar_bd.sql
Query OK, 5 rows affected (0.01 sec)
Records: 5  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 25 rows affected (0.00 sec)
Records: 25  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 150000 rows affected (2.62 sec)
Records: 150000  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 1500000 rows affected (5 min 5.99 sec)
Records: 1500000  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 10000 rows affected (0.23 sec)
Records: 10000  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 200000 rows affected (2.27 sec)
Records: 200000  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 800000 rows affected (1 min 13.40 sec)
Records: 800000  Deleted: 0  Skipped: 0  Warnings: 0

Query OK, 6001215 rows affected (1 hour 21 min 29.62 sec)
Records: 6001215  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 3.7. Llenado de las tablas en la BD TPC-H en MySQL

Una vez generada la BD original, se copió a cada una de las BD donde se realizaron los distintos tipos de particionamiento mencionados anteriormente. Esto se logró haciendo uso de la terminal de Windows, haciendo uso del comando "mysqldumb" y "mysql", como se observa en la Figura 3.8. 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 otra BD el cual se recibe con el comando mysql. Además, para la ejecución de este comando es necesario indicar directamente la contraseña de usuario de MySQL. Este proceso

se realizó en cada una de las bases de datos para implementar los distintos tipos de particionamiento mencionados anteriormente.

```
C:\Users\QO FAMILY>mysqldump -u root -p5643 tpch | mysql -u root -p5643 tpch_1
mysqldump: [Warning] Using a password on the command line interface can be insecure.
mysql: [Warning] Using a password on the command line interface can be insecure.

C:\Users\QO FAMILY>
```

Figura 3.8. Ejemplo de importación de la BD TPC-H original a la fragmentada en MySQL

3.4.1.3. Fragmentación 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). Para lo cual se generó un *script* para cada tipo de partición.

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, para MySQL esto fue una limitación y por ende se desarrolló el *script* que elimina las llaves foráneas de las tablas a las cuales se realizó el particionamiento y aquellas con las que tenían relación y esto se aplicó en cada BD antes de ser particionada. La Figura 3.9 muestra el *script* que elimina las llaves foráneas de la BD TPC-H.

```
-- Eliminar llaves foraneas
-- NATION
ALTER TABLE NATION DROP FOREIGN KEY NATION_FK1;

-- SUPPLIER
ALTER TABLE SUPPLIER DROP FOREIGN KEY SUPPLIER_FK1;

-- CUSTOMER
ALTER TABLE CUSTOMER DROP FOREIGN KEY CUSTOMER_FK1;

-- PARTSUPP
ALTER TABLE PARTSUPP DROP FOREIGN KEY PARTSUPP_FK1, DROP FOREIGN KEY PARTSUPP_FK2;

-- ORDERS
ALTER TABLE ORDERS DROP FOREIGN KEY ORDERS_FK1;

-- LINEITEM
ALTER TABLE LINEITEM DROP FOREIGN KEY LINEITEM_FK1, DROP FOREIGN KEY LINEITEM_FK2;
```

Figura 3.9. *Script* que elimina las llaves foráneas de la BD TPC-H en MySQL

Por otra parte, en MySQL, cuando se genera el particionamiento, el SGBD crea un espacio de tabla a la cual solo se accede haciendo referencia a la tabla principal y no es visible de forma global. Por ejemplo, la siguiente consulta a una partición llamada EMP1 que es una partición de la tabla EMP, es válida en MySQL: "SELECT * FROM EMP PARTITION (EMP1)".

3.4.1.4. Particionamiento Range en la BD TPC-H

Como ya se mencionó anteriormente, el método de partición Range asigna filas a las particiones en función de los valores de columna (atributo) que se encuentran dentro de un rango determinado, por lo que una consulta en donde se obtiene el menor tiempo de respuesta posible es aquella que realiza la búsqueda sobre una sola partición, esto se conoce como consulta dirigida. Por su parte, las consultas de TPC-H, son consultas complejas con múltiples filtros y operaciones de agregación (suma, promedio, mínimo, máximo) sobre distintas tablas a la vez. Ante este contexto, resulta difícil realizar un particionamiento Range adecuado para cada tabla, ya que cada consulta filtra los datos mediante distintos atributos de las tablas. Sin embargo, los atributos más usados en las consultas son las llaves primarias.

Para este proceso, se seleccionaron las cinco tablas más pobladas de las ocho que proporciona el *benchmark*, ya que en las tablas menos pobladas el hecho de aplicar algún tipo de particionamiento no representa una mejora significativa. Las tablas particionadas fueron PART, PARTSUPP, CUSTOMER, ORDERS Y LINEITEM, todas particionadas a través de su llave primaria. En la Figura 3.10 se muestra un ejemplo de particionamiento Range a la tabla PART de la BD TPC-H. Esta tabla cuenta con 200000 registros con un rango de valores de 1 a 200000 para la columna que es llave primaria, y dado que un patrón común en las consultas es, que incluyen la llave primaria, se optó por dividir la tabla de forma arbitraria en 10 fragmentos, con el objetivo de dividir la tabla en particiones que den como resultado una distribución equitativa y así lograr agilizar la búsqueda. La misma idea se siguió para particionar las otras tablas.

```
-- 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
);
```

Figura 3.10. Ejemplo de particionamiento Range en la tabla PART de TPC-H en MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpch_range como se observa en la Figura 3.11, quedando las tablas particionadas de la siguiente manera:

- PART: se generaron 10 particiones con 20,000 tuplas cada una.
- PARTSUPP: se generaron 8 particiones con 100,000 tuplas cada una.
- CUSTOMER: se generaron 5 particiones con 30,000 tuplas cada una.
- ORDERS: se generaron 6 particiones con 250,000 tuplas cada una.
- LINEITEM: se generaron 12 particiones con una cantidad de entre 499,183 y 500,918 tuplas cada una.

```
mysql> source D:\Documentos\Suriel\Tesis\TPC-H\MySQL\range.sql
Query OK, 200000 rows affected (1.68 sec)
Records: 200000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.32 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 800000 rows affected (8.33 sec)
Records: 800000  Duplicates: 0  Warnings: 0

Query OK, 150000 rows affected (1.76 sec)
Records: 150000  Duplicates: 0  Warnings: 0

Query OK, 1500000 rows affected (16.36 sec)
Records: 1500000  Duplicates: 0  Warnings: 0

Query OK, 6001215 rows affected (1 min 21.04 sec)
Records: 6001215  Duplicates: 0  Warnings: 0
```

Figura 3.11. Ejecución del particionamiento Range en la BD TPC-H en MySQL

3.4.1.5. Particionamiento List en la BD TPC-H

Como ya se mencionó anteriormente, en el método List la partición se selecciona en función de columnas que coincidan con una de un conjunto de valores discretos. Ante este contexto, las columnas de las tablas PART y PARTSUPP, contienen un amplio rango de valores, siendo la mayoría valores no discretos. Por tal motivo, en este caso solo fueron tres tablas a las que se le aplicó el particionamiento List.

Se desarrolló el *script* que realiza el particionamiento a las tablas CUSTOMER, ORDERS y LINEITEM, la primera a través de su atributo C_NATIONKEY, logrando así generar particiones por región (África, América, Asia, Europa y Medio Oriente), la segunda por medio de su atributo O_ORDERDATE, obteniendo así particiones por año de orden y la tercera, fue a través del atributo L_SHIPDATE, para generar particiones por año de compra. La Figura 3.12 muestra un ejemplo de particionamiento List en la tabla CUSTOMER de la BD TPC-H.

```
-- 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)
);
```

Figura 3.12. Ejemplo de particionamiento List en la tabla CUSTOMER de TPC-H en MySQL

En MySQL, cuando una tabla contiene llaves primarias, la columna a través de la cual se realice el particionamiento es necesario que pertenezca a ese conjunto de índices. Por tal motivo, se optó por eliminar dichas restricciones en el caso del particionamiento List, tal como se muestra en la Figura 3.13, ya que en caso contrario habría que modificar las relaciones para esas tablas, convirtiendo los atributos utilizados en llaves primarias.

```
-- Eliminar llaves primarias
ALTER TABLE CUSTOMER DROP PRIMARY KEY;
ALTER TABLE ORDERS DROP PRIMARY KEY;
ALTER TABLE LINEITEM DROP PRIMARY KEY;
```

Figura 3.13. Eliminación de llaves primarias en la BD TPC-H en MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpch_list, tal como se observa en la Figura 3.14, quedando las tablas particionadas de la siguiente manera:

- CUSTOMER: se generaron 5 particiones con una cantidad de entre 29,764 y 30,197 tuplas cada una.
- ORDERS: se generaron 7 particiones con una cantidad de entre 133,623 y 228,637 tuplas cada una.
- LINEITEM: se generaron 7 particiones con una cantidad de entre 686,842 y 914,963 tuplas cada una.

```
mysql> source D:\Documentos\Suriel\Tesis\TPC-H\MySQL\list.sql
Query OK, 150000 rows affected (1.66 sec)
Records: 150000  Duplicates: 0  Warnings: 0

Query OK, 1500000 rows affected (18.54 sec)
Records: 1500000  Duplicates: 0  Warnings: 0

Query OK, 6001215 rows affected (5 min 9.32 sec)
Records: 6001215  Duplicates: 0  Warnings: 0
```

Figura 3.14. Ejecución del particionamiento List en la BD TPC-H en MySQL

3.4.1.6. Particionamiento Hash en la BD TPC-H

Continuando con el particionamiento Hash, en este caso se particionaron las mismas tablas que las particionadas con Range y de la misma manera, se llevó a cabo a través de la llave primaria. La Figura 3.15 muestra el *script* generado para particionar las tablas por el método Hash.

```
-- 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;
```

Figura 3.15. Particionamiento Hash de la BD TPC-H en MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpch_hash, tal como se observa en la Figura 3.16, quedando las tablas particionadas de la siguiente manera:

- PART: se generaron 10 particiones con 20,000 tuplas cada una.
- PARTSUPP: se generaron 8 particiones con 100,000 tuplas cada una.
- CUSTOMER: se generaron 5 particiones con 30,000 tuplas cada una.
- ORDERS: se generaron 6 particiones con 250,000 tuplas cada una.
- LINEITEM: se generaron 12 particiones con una cantidad de entre 498,605 y 501,248 tuplas cada una.

Este tipo de particionamiento busca dividir de forma equitativa las particiones, por lo que la mayoría de las tablas en la BD TPC-H quedan con la misma cantidad de tuplas.

```
mysql> source D:\Documentos\Suriel\Tesis\TPC-H\MySQL\hash.sql
Query OK, 200000 rows affected (1.71 sec)
Records: 200000  Duplicates: 0  Warnings: 0

Query OK, 800000 rows affected (14.09 sec)
Records: 800000  Duplicates: 0  Warnings: 0

Query OK, 150000 rows affected (2.44 sec)
Records: 150000  Duplicates: 0  Warnings: 0

Query OK, 1500000 rows affected (28.08 sec)
Records: 1500000  Duplicates: 0  Warnings: 0

Query OK, 6001215 rows affected (8 min 29.66 sec)
Records: 6001215  Duplicates: 0  Warnings: 0
```

Figura 3.16. Ejecución del particionamiento Hash en la BD TPC-H en MySQL

3.4.1.7. Particionamiento Key en la BD TPC-H

El particionamiento Key fue similar al Hash, ya que en estos no se definen las particiones a crear, sino que se crean de forma automática indicando solo la cantidad de particiones deseadas. Además, al igual que el particionamiento Range y Hash se particionaron las mismas tablas a través de la llave primaria. La Figura 3.17 muestra el particionamiento de la BD TPC-H por el método Key.

```
-- 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;
```

Figura 3.17. Particionamiento Key de la BD TPC-H en MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpch_key, tal como se observa en la Figura 3.18, quedando las tablas particionadas de la siguiente manera:

- PART: se generaron 10 particiones con una cantidad de entre 19,513 y 20,885 tuplas cada una.
- PARTSUPP: se generaron 8 particiones con una cantidad de entre 35,488 y 189,760 tuplas cada una.
- CUSTOMER: se generaron 5 particiones con una cantidad de entre 28,940 y 30,684 tuplas cada una.
- ORDERS: se generaron 6 particiones con una cantidad de entre 187,686 y 314,744 tuplas cada una.
- LINEITEM: se generaron 12 particiones con una cantidad de entre 362,131 y 764,024 tuplas cada una.

En la documentación de MySQL no se menciona de forma exacta cómo se distribuyen los datos a través del método de particionamiento Key, y tampoco se

profundizó en indagar más sobre ello, ya que no era el objetivo primordial de esta tesis. Por tanto, fue necesario obtener el dato con la función "count()" de MySQL. Lo mismo se hizo en los demás tipos de particionamiento en donde también se desconocía el dato exacto de la cantidad de tuplas para cada tabla.

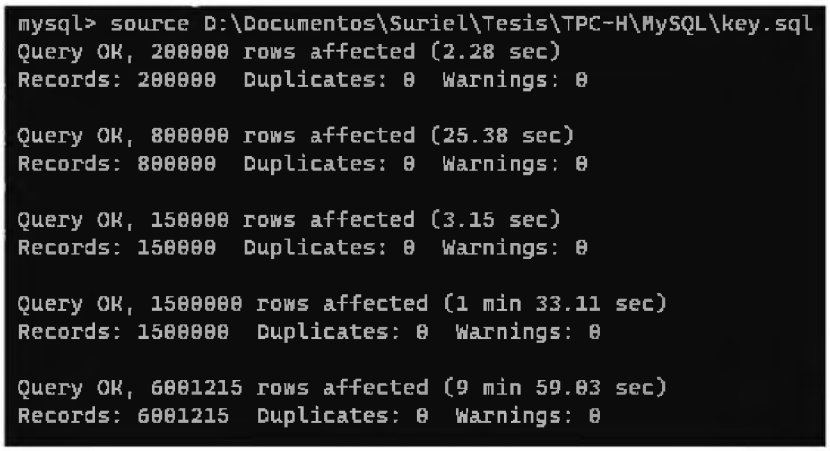


Figura 3.18. Ejecución del particionamiento Key en la BD TPC-H en MySQL

3.4.1.8. Generación y llenado de la BD TPC-E original

Para la generación de la BD TPC-E se siguió la misma dinámica que en el caso de TPC-H. Para ello, se creó en primera instancia una BD nombrada "tpce", seguido de cuatro bases de datos más, a las cuales se les aplicaría cada tipo de particionamiento. Los siguientes nombres serán utilizados para referirse a las BD TPC-E en la etapa de implementación.

- **tpce**: base de datos original de TPC-E
- **tpce_range**: base de datos para el particionamiento Range
- **tpce_list**: base de datos para el particionamiento List
- **tpce_hash**: base de datos para el particionamiento Hash
- **tpce_key**: base de datos para el particionamiento Key

Teniendo los esquemas de BD creados, se pasó a realizar directamente el llenado de cada una de las BD, ya que, para este caso, ya se contaba con una copia de TPC-E como se muestra en la Figura 3.19, que es un fragmento del *script* que crea dicha BD, el cual fue proporcionado por el Maestro en Sistemas Computacionales Felipe Castro Medina, quien trabajó con dicho *benchmark* en su

tesis titulada "APLICACIÓN DE MÉTODOS DE FRAGMENTACIÓN Y REPLICACIÓN DE DATOS EN LA NUBE" (Medina, 2019). Por tal motivo, se pasó directamente al llenado de la BD TPC-E original y posteriormente se copió a cada uno de los esquemas creados para los distintos tipos de particionamiento. La Figura 3.20 muestra la ejecución del *script* que crea la BD TPC-E.

```
--
-- Estructura de tabla para la tabla `ACCOUNT_PERMISSION`
--

CREATE TABLE `ACCOUNT_PERMISSION` (
  `AP_CAT_ID` bigint(11) NOT NULL,
  `AP_ACL` varchar(4) NOT NULL,
  `AP_TAX_ID` varchar(20) NOT NULL,
  `AP_L_NAME` varchar(25) NOT NULL,
  `AP_F_NAME` varchar(20) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Volcado de datos para la tabla `ACCOUNT_PERMISSION`
--

INSERT INTO `ACCOUNT_PERMISSION` (`AP_CAT_ID`, `AP_ACL`, `AP_TAX_ID`, `AP_L_NAME`, `AP_F_NAME`) VALUES
(43000000001, '0', '078GO5457DB627', 'Fowle', 'Joshua'),
(43000000002, '0', '078GO5457DB627', 'Fowle', 'Joshua'),
(43000000002, '1', '240HU9704ZL947', 'Peraro', 'Patrick'),
(43000000003, '0', '078GO5457DB627', 'Fowle', 'Joshua'),
(43000000004, '0', '078GO5457DB627', 'Fowle', 'Joshua'),
(43000000005, '0', '078GO5457DB627', 'Fowle', 'Joshua'),
(43000000005, '1', '449OP9662AA863', 'Kemp', 'Jose'),
(43000000006, '0', '078GO5457DB627', 'Fowle', 'Joshua'),
(43000000006, '1', '902GT0960WJ546', 'Valderas', 'Maryann'),
```

Figura 3.19. Fragmento del *script* que crea la BD TPC-E en MySQL

```
mysql> source D:\Documentos\Tesis\TPC-E\BackTPCE.sql
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.07 sec)
```

Figura 3.20. Creación de la BD TPC-E en MySQL

3.4.1.9. Fragmentación de la BD TPC-E

Teniendo las bases de datos creadas, se pasó al desarrollo de cada tipo de particionamiento que admite MySQL (Range, List, Hash y Key) y se generó un *script* para cada tipo de particionamiento respectivamente. Además, también fue necesario eliminar las llaves foráneas al igual que en TPC-H.

3.4.1.10. Particionamiento Range en la BD TPC-E

En este caso, se particionaron cuatro tablas: TRADE, SETTLEMENT, TRADE_HISTORY y HOLDING_HISTORY, todas particionadas a través de su llave primaria. En la Figura 3.21 se presenta un ejemplo de particionamiento Range en la tabla TRADE de la BD TPC-E.

```
-- For table TRADE(1728000)
ALTER TABLE TRADE PARTITION BY RANGE (T_ID) (
        PARTITION T1 VALUES LESS THAN (200000000174528),
  PARTITION T2 VALUES LESS THAN (200000000349056),
  PARTITION T3 VALUES LESS THAN (200000000523584),
  PARTITION T4 VALUES LESS THAN (200000000698112),
  PARTITION T5 VALUES LESS THAN (200000000872640),
  PARTITION T6 VALUES LESS THAN (200000001047168),
  PARTITION T7 VALUES LESS THAN (200000001221696),
  PARTITION T8 VALUES LESS THAN (200000001396224),
  PARTITION T9 VALUES LESS THAN (200000001570752),
  PARTITION T10 VALUES LESS THAN MAXVALUE
);
```

Figura 3.21. Ejemplo de particionamiento Range en la tabla TRADE de TPC-E en MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpce_range, como se muestra en la Figura 3.22, quedando las tablas particionadas de la siguiente manera:

- TRADE: se generaron 10 particiones con 172,800 tuplas cada una.
- SETTLEMENT: se generaron 10 particiones con 172,800 tuplas cada una.
- TRADE_HISTORY: se generaron 10 particiones con una cantidad de entre 414,588 y 415,073 tuplas cada una.
- HOLDING_HISTORY: se generaron 10 particiones con una cantidad de entre 125,631 y 338,472 tuplas cada una.

```
mysql> source D:\Documentos\Suriel\Tesis\TPC-E\MySQL\range.sql
Query OK, 1728000 rows affected (30.37 sec)
Records: 1728000  Duplicates: 0  Warnings: 0

Query OK, 4148112 rows affected (28.91 sec)
Records: 4148112  Duplicates: 0  Warnings: 0

Query OK, 1728000 rows affected (8.51 sec)
Records: 1728000  Duplicates: 0  Warnings: 0

Query OK, 2247265 rows affected (22.11 sec)
Records: 2247265  Duplicates: 0  Warnings: 0
```

Figura 3.22. Ejecución del particionamiento Range en la BD TPC-E en MySQL

3.4.1.11. Particionamiento List en la BD TPC-E

Para este caso, solo se particionaron dos tablas que eran las que contaban con los atributos y características necesarias para poder realizar este tipo de particionamiento. Las tablas particionadas fueron TRADE_HISTORY y HOLDING_HISTORY, la primera se particionó a través de su atributo TH_DTS, logrando así generar particiones por días del mes, la segunda se particionó a través de su atributo HH_AFTER_QTY, el cual tuvo valores que iban desde -800 a 800 en intervalos de 100. La Figura 3.23 muestra un ejemplo de particionamiento List en la tabla TRADE_HISTORY de la BD TPC-E.

```
-- For table TRADE_HISTORY (4148112)
-- Se realizó particionamiento por día
-- a través del atributo TH_DTS
ALTER TABLE TRADE_HISTORY PARTITION BY LIST (DAY(TH_DTS)) (
        PARTITION TH1 VALUES IN (1,2,3),
   PARTITION TH2 VALUES IN (4,5,6),
   PARTITION TH3 VALUES IN (7,8,9),
   PARTITION TH4 VALUES IN (10,11,12),
   PARTITION TH5 VALUES IN (13,14,15),
   PARTITION TH6 VALUES IN (16,17,18),
   PARTITION TH7 VALUES IN (19,20,21),
   PARTITION TH8 VALUES IN (22,23,24),
   PARTITION TH9 VALUES IN (25,26,27),
   PARTITION TH10 VALUES IN (28,29,30,31)
);
```

Figura 3.23. Ejemplo de particionamiento List en la tabla TRADE_HISTORY de TPC-E en MySQL

Además, es importante mencionar que al igual que en el particionamiento List con TPC-H, además de eliminar las llaves foráneas, también fue necesario eliminar las llaves primarias de aquellas tablas particionadas, como se observa en la Figura 3.24, ya que los atributos utilizados para realizar este tipo de particionamiento no

son llaves primarias y el hecho de tener llave primaria imposibilita poder realizar particionamiento a otros atributos.

```
mysql> ALTER TABLE TRADE_HISTORY DROP PRIMARY KEY;
Query OK, 4148112 rows affected (26.55 sec)
Records: 4148112  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE HOLDING_HISTORY DROP PRIMARY KEY;
Query OK, 2247265 rows affected (19.72 sec)
Records: 2247265  Duplicates: 0  Warnings: 0
```

Figura 3.24. Eliminación de llaves primarias en la BD TPC-E con MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpce_list, tal como se muestra en la Figura 3.25, quedando las tablas particionadas de la siguiente manera:

- TRADE_HISTORY: se generaron 10 particiones con una cantidad de entre 138,215 y 691,288 tuplas cada una.
- HOLDING_HISTORY: se generaron 6 particiones con una cantidad de entre 140,193 y 1,095,746 tuplas cada una.

```
mysql> source D:\Documentos\Suriel\Tesis\TPC-E\MySQL\list.sql
Query OK, 4148112 rows affected (32.60 sec)
Records: 4148112  Duplicates: 0  Warnings: 0

Query OK, 2247265 rows affected (28.55 sec)
Records: 2247265  Duplicates: 0  Warnings: 0
```

Figura 3.25. Ejecución del particionamiento List en la BD TPC-E en MySQL

3.4.1.12. Particionamiento Hash en la BD TPC-E

Para este caso, se particionaron cuatro tablas al igual que en el particionamiento Range. Las tablas particionadas fueron TRADE, SETTLEMENT, TRADE_HISTORY y HOLDING_HISTORY, todas particionadas a través de su llave primaria. La Figura 3.26 muestra el particionamiento de la BD TPC-E por el método Hash.

```
-- For table TRADE(1728000)
ALTER TABLE TRADE PARTITION BY HASH (T_ID) PARTITIONS 10;

-- For table TRADE_HISTORY (4148112)
ALTER TABLE TRADE_HISTORY PARTITION BY HASH (TH_T_ID) PARTITIONS 10;

-- For table SETTLEMENT (1728000)
ALTER TABLE SETTLEMENT PARTITION BY HASH (SE_T_ID) PARTITIONS 10;

-- For table HOLDING_HISTORY (2247265)
ALTER TABLE HOLDING_HISTORY PARTITION BY HASH (HH_H_T_ID) PARTITIONS 10;
```

Figura 3.26. Particionamiento Hash de la BD TPC-E en MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpce_hash, tal como se muestra en la Figura 3.27, quedando las tablas particionadas de la siguiente manera:

- TRADE: se generaron 10 particiones con 172,800 tuplas cada una.
- SETTLEMENT: se generaron 10 particiones con 172,800 tuplas cada una.
- TRADE_HISTORY: se generaron 10 particiones con una cantidad de entre 414,451 y 415,361 tuplas cada una.
- HOLDING_HISTORY: se generaron 10 particiones con una cantidad de entre 223,968 y 225,535 tuplas cada una.

```
mysql> source D:\Documentos\Suriel\Tesis\TPC-E\MySQL\hash.sql
Query OK, 1728000 rows affected (3 min 14.96 sec)
Records: 1728000  Duplicates: 0  Warnings: 0

Query OK, 4148112 rows affected (33.92 sec)
Records: 4148112  Duplicates: 0  Warnings: 0

Query OK, 1728000 rows affected (8.70 sec)
Records: 1728000  Duplicates: 0  Warnings: 0

Query OK, 2247265 rows affected (24.70 sec)
Records: 2247265  Duplicates: 0  Warnings: 0
```

Figura 3.27. Ejecución del particionamiento Hash en la BD TPC-E en MySQL

3.4.1.13. Particionamiento Key en la BD TPC-E

Al igual que en el particionamiento Hash, las tablas particionadas fueron TRADE, SETTLEMENT, TRADE_HISTORY y HOLDING_HISTORY, todas particionadas a través de su llave primaria. La Figura 3.28 muestra el particionamiento de la BD TPC-E por el método Key.

```
-- For table TRADE(1728000)
ALTER TABLE TRADE PARTITION BY KEY (T_ID) PARTITIONS 10;

-- For table TRADE_HISTORY (4148112)
ALTER TABLE TRADE_HISTORY PARTITION BY KEY (TH_T_ID) PARTITIONS 10;

-- For table SETTLEMENT (1728000)
ALTER TABLE SETTLEMENT PARTITION BY KEY (SE_T_ID) PARTITIONS 10;

-- For table HOLDING_HISTORY (2247265)
ALTER TABLE HOLDING_HISTORY PARTITION BY KEY (HH_H_T_ID) PARTITIONS 10;
```

Figura 3.28. Particionamiento Key de la BD TPC-E en MySQL

Teniendo el *script* generado, este se ejecutó en la BD tpce_key, tal como se muestra en la Figura 3.29, quedando las tablas particionadas de la siguiente manera:

- TRADE: se generaron 10 particiones con una cantidad de entre 341,720 y 352,215 tuplas cada una.
- SETTLEMENT: se generaron 10 particiones con una cantidad de entre 340,834 y 351,525 tuplas cada una.
- TRADE_HISTORY: se generaron 10 particiones con una cantidad de entre 820,616 y 845,633 tuplas cada una.
- HOLDING_HISTORY: se generaron 10 particiones con una cantidad de entre 443,543 y 458,137 tuplas cada una.

```
mysql> source D:\Documentos\Suriel\Tesis\TPC-E\MySQL\key.sql
Query OK, 1728000 rows affected (3 min 27.59 sec)
Records: 1728000  Duplicates: 0  Warnings: 0

Query OK, 4148112 rows affected (27.26 sec)
Records: 4148112  Duplicates: 0  Warnings: 0

Query OK, 1728000 rows affected (8.19 sec)
Records: 1728000  Duplicates: 0  Warnings: 0

Query OK, 2247265 rows affected (20.18 sec)
Records: 2247265  Duplicates: 0  Warnings: 0
```

Figura 3.29. Ejecución del particionamiento Key en la BD TPC-E en MySQL

3.4.2. PostgreSQL

3.4.2.1. Instalación y configuraciones previas

Siguiendo con PostgreSQL, este se descargó también de su Web oficial (PostgreSQL Downloads, 2022), específicamente la versión 14.1. Esta versión es la más actual a la fecha en que se realiza esta implementación y es la versión gratuita bajo los términos de licencia PostgreSQL, que es una licencia libre y de código abierto, similar a las licencias BSD o MIT. Además del servidor de PostgreSQL, el paquete de instalación cuenta con una herramienta visual para su uso llamada pgAdmin, la cual se encuentra en su versión 4.

Posteriormente, se instaló PostgreSQL en el equipo donde se realizaron las pruebas y al igual que con MySQL, fue necesario agregar a las variables de entorno de Windows, la ruta donde se encuentran los archivos ejecutables de PostgreSQL, para de igual forma poder interactuar con el servidor desde la propia terminal de Windows.

3.4.2.2. Generación y llenado de la BD TPC-H original

En este caso la dinámica fue un poco distinta a MySQL. Primeramente, se creó la BD original de TPC-H utilizando el comando “create database” para crear la BD. Posteriormente, se generaron tres bases de datos más y se implementó en ellas los tipos de particionamiento admitidos por PostgreSQL.

Teniendo los esquemas de BD creados, se pasó a crear las tablas en la BD original y en el caso de PostgreSQL, tampoco fue necesario modificar el *script* que crea las tablas. El *script* se ejecutó haciendo uso de la sentencia “\i” de PostgreSQL, que permite ejecutar órdenes desde archivos SQL, tal como se observa en la Figura 3.30.

```
postgres=# create database tpch;
CREATE DATABASE
postgres=# \c tpch
Ahora está conectado a la base de datos «tpch» con el usuario «postgres».
tpch=# \i D:/Documentos/Tesis/TPC-H/PostgreSQL/dss.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.30. Creación de las tablas en la BD TPC-H en PostgreSQL

Teniendo las tablas creadas, se agregaron las relaciones utilizando el mismo *script* que en MySQL. En la Figura 3.31 se muestra la ejecución del *script* que crea las relaciones en la BD TPC-H.

```
tpch=# \i D:/Documentos/Tesis/TPC-H/PostgreSQL/dss.ri
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

Figura 3.31. Creación de relaciones en la BD TPC-H en PostgreSQL

Una vez generada la BD junto con las tablas y las relaciones, se realizó el llenado de la BD haciendo uso del comando "copy" de PostgreSQL. Además, al igual que en MySQL, se desarrolló un pequeño *script* que automatiza el proceso de llenado de las tablas, como se describe en la Figura 3.32. Además, en la Figura 3.33 se muestra la ejecución del *script* anterior.

```
copy region from 'C:/tpch/region.tbl' with delimiter as '|';
copy nation from 'C:/tpch/nation.tbl' with delimiter as '|';
copy customer from 'C:/tpch/customer.tbl' with delimiter as '|';
copy orders from 'C:/tpch/orders.tbl' with delimiter as '|';
copy supplier from 'C:/tpch/supplier.tbl' with delimiter as '|';
copy part from 'C:/tpch/part.tbl' with delimiter as '|';
copy partsupp from 'C:/tpch/partsupp.tbl' with delimiter as '|';
copy lineitem from 'C:/tpch/lineitem.tbl' with delimiter as '|';
```

Figura 3.32. *Script* que carga las tuplas en la BD TPC-H en PostgreSQL

```
tpch=# \i D:/Documentos/Tesis/TPC-H/PostgreSQL/load.sql
COPY 5
COPY 25
COPY 150000
COPY 1500000
COPY 10000
COPY 200000
COPY 800000
COPY 6001215
```

Figura 3.33. Llenado de las tablas en la BD TPC-H en PostgreSQL

Teniendo la BD original generada, se pasó a la creación de las tablas en las BD a particionar. En PostgreSQL es necesario especificar desde la creación de las tablas el tipo de particionamiento a emplear y no es posible alterar las tablas luego de ser creadas, a diferencia de MySQL en donde sí es posible modificarlas. Por tal motivo, se creó un *script* por cada tipo de particionamiento para la creación de las tablas, definiendo en cada uno de ellos el tipo de particionamiento que se utilizó. Las Figuras 3.34, 3.36 y 3.38 muestran un fragmento del *script* que crea las tablas de la BD TPC-H acorde a cada tipo de particionamiento.

```
CREATE TABLE CUSTOMER ( C_CUSTKEY     INTEGER NOT NULL,
                        C_NAME        VARCHAR(25) NOT NULL,
                        C_ADDRESS     VARCHAR(40) NOT NULL,
                        C_NATIONKEY   INTEGER NOT NULL,
                        C_PHONE       CHAR(15) NOT NULL,
                        C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
                        C_MKTSEGMENT  CHAR(10) NOT NULL,
                        C_COMMENT     VARCHAR(117) NOT NULL
) PARTITION BY RANGE (C_CUSTKEY);

CREATE TABLE ORDERS  ( O_ORDERKEY       INTEGER NOT NULL,
                       O_CUSTKEY        INTEGER NOT NULL,
                       O_ORDERSTATUS    CHAR(1) NOT NULL,
                       O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
                       O_ORDERDATE      DATE NOT NULL,
                       O_ORDERPRIORITY  CHAR(15) NOT NULL,
                       O_CLERK          CHAR(15) NOT NULL,
                       O_SHIPPRIORITY   INTEGER NOT NULL,
                       O_COMMENT        VARCHAR(79) NOT NULL
) PARTITION BY RANGE (O_ORDERKEY);
```

Figura 3.34. Fragmento del *script* que crea las tablas de la BD TPC-H con particionamiento Range en PostgreSQL

Posteriormente, se agregaron las relaciones a cada BD particionada, como se observa en las Figuras 3.35, 3.37 y 3.39.

```
tpch_range=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/particiones/range.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
tpch_range=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/dss.ri
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

Figura 3.35. Creación de tablas y relaciones en la BD TPC-H con particionamiento Range en PostgreSQL

```
CREATE TABLE CUSTOMER ( C_CUSTKEY     INTEGER NOT NULL,
                        C_NAME        VARCHAR(25) NOT NULL,
                        C_ADDRESS     VARCHAR(40) NOT NULL,
                        C_NATIONKEY   INTEGER NOT NULL,
                        C_PHONE       CHAR(15) NOT NULL,
                        C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
                        C_MKTSEGMENT  CHAR(10) NOT NULL,
                        C_COMMENT     VARCHAR(117) NOT NULL
) PARTITION BY LIST (C_NATIONKEY);

CREATE TABLE ORDERS  ( O_ORDERKEY       INTEGER NOT NULL,
                       O_CUSTKEY        INTEGER NOT NULL,
                       O_ORDERSTATUS    CHAR(1) NOT NULL,
                       O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
                       O_ORDERDATE      DATE NOT NULL,
                       O_ORDERPRIORITY  CHAR(15) NOT NULL,
                       O_CLERK          CHAR(15) NOT NULL,
                       O_SHIPPRIORITY   INTEGER NOT NULL,
                       O_COMMENT        VARCHAR(79) NOT NULL
) PARTITION BY LIST (EXTRACT(YEAR FROM O_ORDERDATE));
```

Figura 3.36. Fragmento del *script* que crea las tablas de la BD TPC-H con particionamiento List en PostgreSQL

Para este caso en particular, al igual que sucedió con MySQL fue necesario descartar tanto llaves primarias como foráneas para realizar el particionamiento List, por lo que se desarrolló el *script* que crea las relaciones descartando en él, las llaves primarias y foráneas.

```
tpch_list=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/particiones/list.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
tpch_list=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/particiones/list.ri
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

Figura 3.37. Creación de tablas y relaciones en la BD TPC-H con particionamiento List en PostgreSQL

```
CREATE TABLE CUSTOMER ( C_CUSTKEY     INTEGER NOT NULL,
                        C_NAME        VARCHAR(25) NOT NULL,
                        C_ADDRESS     VARCHAR(40) NOT NULL,
                        C_NATIONKEY   INTEGER NOT NULL,
                        C_PHONE       CHAR(15) NOT NULL,
                        C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
                        C_MKTSEGMENT  CHAR(10) NOT NULL,
                        C_COMMENT     VARCHAR(117) NOT NULL
) PARTITION BY HASH (C_CUSTKEY);

CREATE TABLE ORDERS  ( O_ORDERKEY       INTEGER NOT NULL,
                       O_CUSTKEY        INTEGER NOT NULL,
                       O_ORDERSTATUS    CHAR(1) NOT NULL,
                       O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
                       O_ORDERDATE      DATE NOT NULL,
                       O_ORDERPRIORITY  CHAR(15) NOT NULL,
                       O_CLERK          CHAR(15) NOT NULL,
                       O_SHIPPRIORITY   INTEGER NOT NULL,
                       O_COMMENT        VARCHAR(79) NOT NULL
) PARTITION BY HASH (O_ORDERKEY);
```

Figura 3.38. Fragmento del *script* que crea las tablas de la BD TPC-H con particionamiento Hash en PostgreSQL

```
tpch_hash=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/particiones/hash.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
tpch_hash=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/dss.ri
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

Figura 3.39. Creación de tablas y relaciones en la BD TPC-H con particionamiento Hash en PostgreSQL

Además, se realizaron los métodos de particionamiento descritos a continuación y por último se agregaron los datos a cada una de las bases de datos particionadas. En PostgreSQL se realizó de esta manera, ya que así lo define la documentación y es como se suele realizar, ya que es una limitante que algunos SGBD tienen, al no poder alterar una tabla para definir el particionamiento, luego de ser creada.

3.4.2.3. Fragmentación de la BD TPC-H

Teniendo las bases de datos creadas, se pasó al desarrollo de cada tipo de particionamiento que admite PostgreSQL (Range, List y Hash). Para lo cual se generó un *script* para cada tipo de partición.

En PostgreSQL no hubo problema alguno al realizar el particionamiento sobre índices que son llave foránea y, por lo tanto, no fue necesario eliminar estas restricciones para el particionamiento Range y Hash.

Por otra parte, en PostgreSQL, cuando se genera el particionamiento, el SGBD crea una tabla a la cual se accede fácilmente de manera global, como si de una tabla normal se tratase, algo que MySQL maneja de forma diferente. Por ejemplo, la siguiente consulta a una partición llamada EMP1 que es una partición de la tabla EMP, es válida en PostgreSQL: "SELECT * FROM EMP1", pero no es válida en MySQL.

3.4.2.4. Particionamiento Range en la BD TPC-H

Para este proceso, se seleccionaron las mismas tablas que en el caso de MySQL: PART, PARTSUPP, CUSTOMER, ORDERS Y LINEITEM. La Figura 3.40 muestra un ejemplo de particionamiento Range en la tabla ORDERS de la BD TPC-H.

```
-- For table ORDERS(1500000)
CREATE TABLE O1 PARTITION OF ORDERS FOR VALUES FROM (MINVALUE) TO (1000001);
CREATE TABLE O2 PARTITION OF ORDERS FOR VALUES FROM (1000001) TO (2000001);
CREATE TABLE O3 PARTITION OF ORDERS FOR VALUES FROM (2000001) TO (3000001);
CREATE TABLE O4 PARTITION OF ORDERS FOR VALUES FROM (3000001) TO (4000001);
CREATE TABLE O5 PARTITION OF ORDERS FOR VALUES FROM (4000001) TO (5000001);
CREATE TABLE O6 PARTITION OF ORDERS FOR VALUES FROM (5000001) TO (MAXVALUE);
```

Figura 3.40. Ejemplo de particionamiento Range en la tabla ORDERS de TPC-H en PostgreSQL

Además, en la Figura 3.41 se muestra la ejecución del *script* que genera el particionamiento Range en la BD TPC-H.

Figura 3.41. Ejecución del particionamiento Range en la BD TPC-H en PostgreSQL

Posteriormente, se cargaron los datos a las tablas de la BD tpch_range con el mismo *script* utilizado para llenar la BD TPC-H original, tal como se muestra en la Figura 3.42, quedando los datos de las particiones distribuidas de la siguiente manera:

- PART: se generaron 10 particiones con 20,000 tuplas cada una.
- PARTSUPP: se generaron 8 particiones con 100,000 tuplas cada una.
- CUSTOMER: se generaron 5 particiones con 30,000 tuplas cada una.
- ORDERS: se generaron 6 particiones con 250,000 tuplas cada una.
- LINEITEM: se generaron 12 particiones con una cantidad de entre 499,183 y 500,918 tuplas cada una.

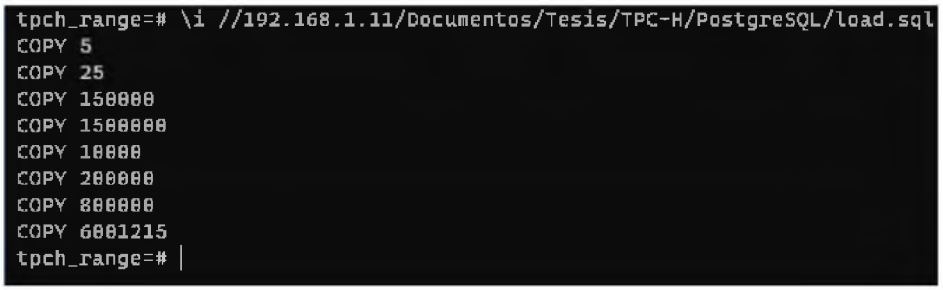


Figura 3.42. Llenado de las tablas en la BD TPC-H con particionamiento Range en PostgreSQL

3.4.2.5. Particionamiento List en la BD TPC-H

De forma similar a MySQL, se particionaron las tablas CUSTOMER, ORDERS y LINEITEM, la primera a través de su atributo C_NATIONKEY, logrando

así generar particiones por región (África, América, Asia, Europa y Medio Oriente), la segunda por medio de su atributo O_ORDERDATE, obteniendo así particiones por año de orden y la tercera, fue a través del atributo L_SHIPDATE, para generar particiones por año de compra. La Figura 3.43 muestra un ejemplo de particionamiento List en la tabla CUSTOMERS de la BD TPC-H.

```
-- For table CUSTOMER(150000)
CREATE TABLE C1 PARTITION OF CUSTOMER FOR VALUES IN (0,5,14,15,16);
CREATE TABLE C2 PARTITION OF CUSTOMER FOR VALUES IN (1,2,3,17,24);
CREATE TABLE C3 PARTITION OF CUSTOMER FOR VALUES IN (8,9,12,18,21);
CREATE TABLE C4 PARTITION OF CUSTOMER FOR VALUES IN (6,7,19,22,23);
CREATE TABLE C5 PARTITION OF CUSTOMER FOR VALUES IN (4,10,11,13,20);
```

Figura 3.43. Ejemplo de particionamiento List en la tabla CUSTOMER de TPC-H en PostgreSQL

Además, para el particionamiento List, sí fue necesario quitar las llaves primarias y foráneas, tal como se hizo en MySQL con este mismo particionamiento, ya que PostgreSQL no permite particionar tablas si la columna utilizada para el particionamiento no pertenece al conjunto de llaves primarias. Estas llaves primarias y foráneas se descartaron en el *script* que se realizó para añadir las relaciones, por lo que no fue necesario eliminarlas en este punto como se hizo en MySQL.

Por su parte, en la Figura 3.44 se muestra la ejecución del *script* que genera el particionamiento List en la BD TPC-H.

```
tpch_list=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/particiones/list.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.44. Ejecución del particionamiento List en la BD TPC-H en PostgreSQL

Posteriormente, se cargaron los datos a las tablas de la BD tpch_list, tal como se muestra en la Figura 3.45, quedando los datos de las particiones distribuidos de la siguiente manera:

- CUSTOMER: se generaron 5 particiones con una cantidad de entre 29,764 y 30,197 tuplas cada una.

- ORDERS: se generaron 7 particiones con una cantidad de entre 133,623 y 228,637 tuplas cada una.
- LINEITEM: se generaron 7 particiones con una cantidad de entre 686,842 y 914,963 tuplas cada una.

Figura 3.45. Llenado de las tablas en la BD TPC-H con particionamiento List en PostgreSQL

3.4.2.6. Particionamiento Hash en la BD TPC-H

Continuando con el particionamiento Hash, se realizaron las mismas particiones que en el caso de MySQL y como se observa en la Figura 3.46, la sintaxis es un poco distinta, ya que, mientras que MySQL genera las particiones de forma automática indicando solo la cantidad de particiones, en PostgreSQL es necesario definir la partición, al igual que en el particionamiento Range y List mencionados anteriormente. Además, se requiere especificar un módulo (cantidad de particiones a crear) y un residuo (cantidad de particiones a crear menos uno).

```
-- For table PARTSUPP(800000)
CREATE TABLE PS1 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 0);
CREATE TABLE PS2 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 1);
CREATE TABLE PS3 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 2);
CREATE TABLE PS4 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 3);
CREATE TABLE PS5 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 4);
CREATE TABLE PS6 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 5);
CREATE TABLE PS7 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 6);
CREATE TABLE PS8 PARTITION OF PARTSUPP FOR VALUES WITH (MODULUS 8, REMAINDER 7);
```

Figura 3.46. Ejemplo de particionamiento Hash en la tabla PARTSUPP de TPC-H en PostgreSQL

Además, en la Figura 3.47 se muestra la ejecución del *script* que genera el particionamiento Hash en la BD TPC-H.

Figura 3.47. Ejecución del particionamiento Hash en la BD TPC-H en PostgreSQL

Posteriormente, se cargaron los datos a las tablas de la BD tpch_hash, tal como se muestra en la Figura 3.48, quedando los datos de las particiones distribuidos de la siguiente manera:

- PART: se generaron 10 particiones con una cantidad de entre 19,766 y 20,199 tuplas cada una.
- PARTSUPP: se generaron 8 particiones con una cantidad de entre 99,180 y 101,140 tuplas cada una.
- CUSTOMER: se generaron 5 particiones con una cantidad de entre 29,874 y 30,198 tuplas cada una.
- ORDERS: se generaron 6 particiones con una cantidad de entre 249,301 y 250,837 tuplas cada una.
- LINEITEM: se generaron 12 particiones con una cantidad de entre 496,765 y 502,951 tuplas cada una.

Este tipo de particionamiento en PostgreSQL, distribuye de forma diferente los datos en las particiones, en donde cada partición contiene las filas para las que el valor Hash de la clave de la partición dividido por el módulo especificado producen el resto especificado, a diferencia de MySQL que busca distribuir los datos de forma equitativa, por lo que fue necesario obtener el dato de la cantidad de tuplas con la función count().

```
tpch_hash=# \i //192.168.1.11/Documentos/Tesis/TPC-H/PostgreSQL/load.sql
COPY 5
COPY 25
COPY 150000
COPY 1500000
COPY 10000
COPY 200000
COPY 800000
COPY 6001215
tpch_hash=#
```

Figura 3.48. Llenado de las tablas en la BD TPC-H con particionamiento Hash en PostgreSQL

3.4.2.7. Generación y llenado de la BD TPC-E original

En este caso la dinámica fue distinta, ya que la copia de la BD con la que se contaba era para MySQL. Primeramente, se empezó creando la BD original, para ello se tomó el *script* que crea las tablas de TPC-E de MySQL, el cual también fue obtenido de (Medina, 2019). Con este *script* se obtuvieron dos archivos, uno para la creación de las tablas de TPC-E y otro que agrega las relaciones de llaves primarias y foráneas. Además, se realizaron las modificaciones pertinentes, tales como el cambio de tipo de dato de las variables para que sean compatibles con PostgreSQL, y se modificó la sintaxis para agregar los índices de las tablas. Esta operación de separar el *script* en dos archivos fue debido a que no se contaba con la jerarquía correcta para llenar las tablas, lo que ocasionaba un error al añadir los datos a las tablas después de crear las relaciones, algo que se solucionó realizando primero el llenado de las tablas y por último agregar las relaciones.

La Figura 3.49 muestra un ejemplo de definición de tabla en la BD TPC-E.

```
--
-- Estructura de tabla para la tabla ACCOUNT_PERMISSION
--

CREATE TABLE ACCOUNT_PERMISSION (
  AP_CAT_ID bigint NOT NULL,
  AP_ACL varchar(4) NOT NULL,
  AP_TAX_ID varchar(20) NOT NULL,
  AP_L_NAME varchar(25) NOT NULL,
  AP_F_NAME varchar(20) NOT NULL
);
```

Figura 3.49. Ejemplo de definición de tabla con TPC-E en PostgreSQL

Teniendo los esquemas de BD creados, se pasó a realizar la creación de las tablas en la BD TPC-E original que consta de 33 tablas, tal como se muestra en la Figura 3.50.

```
postgres=# create database tpce;
CREATE DATABASE
postgres=# \c tpce
Ahora está conectado a la base de datos «tpce» con el usuario «postgres».
tpce=# \i D:/Documentos/Suriel/Tesis/TPC-E/PostgreSQL/dss.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.50. Creación de tablas en la BD TPC-E en PostgreSQL

Una vez generadas las tablas de la BD, se realizó el llenado de las tablas haciendo uso del comando "copy" de PostgreSQL. Además, al igual que en MySQL, se desarrolló un pequeño *script* que automatiza el proceso de llenado de las tablas de TPC-E, como se describe en la Figura 3.51. Así mismo, en la Figura 3.52 se muestra la ejecución de dicho *script*.

```
copy account_permission from 'C:/tpce/AccountPermission.txt' with delimiter as '|';
copy customer from 'C:/tpce/Customer.txt' with delimiter as '|';
copy customer_account from 'C:/tpce/CustomerAccount.txt' with delimiter as '|';
copy customer_taxrate from 'C:/tpce/CustomerTaxrate.txt' with delimiter as '|';
copy holding from 'C:/tpce/Holding.txt' with delimiter as '|';
copy holding_history from 'C:/tpce/HoldingHistory.txt' with delimiter as '|';
copy holding_summary from 'C:/tpce/HoldingSummary.txt' with delimiter as '|';
copy watch_item from 'C:/tpce/WatchItem.txt' with delimiter as '|';
copy watch_list from 'C:/tpce/WatchList.txt' with delimiter as '|';
copy broker from 'C:/tpce/Broker.txt' with delimiter as '|';
copy cash_transaction from 'C:/tpce/CashTransaction.txt' with delimiter as '|';
```

Figura 3.51. *Script* que carga las tuplas en la BD TPC-E en PostgreSQL

```
tpce=# \i D:/Documentos/Suriel/Tesis/TPC-E/PostgreSQL/load.sql
COPY 7128
COPY 1504
COPY 10
COPY 1590458
COPY 15
COPY 240
COPY 500
COPY 1500
COPY 1000
```

Figura 3.52. Llenado de las tablas en la BD TPC-E en PostgreSQL

Po último, se agregaron las relaciones a la BD TPC-E original, como se muestra en la Figura 3.53.

```
tpce=# \i D:/Documentos/Suriel/Tesis/TPC-E/PostgreSQL/dss.ri
ALTER TABLE
ALTER TABLE
CREATE INDEX
ALTER TABLE
CREATE INDEX
ALTER TABLE
ALTER TABLE
ALTER TABLE
CREATE INDEX
CREATE INDEX
```

Figura 3.53. Creación de relaciones en la BD TPC-E en PostgreSQL

Teniendo la BD original generada, se pasó a la creación de las tablas en las BD a particionar. Además, se creó un *script* por cada tipo de particionamiento para la creación de las tablas definiendo en cada uno de ellos el tipo de particionamiento que se utilizó, esto, debido al funcionamiento ya mencionado anteriormente en cuanto a la creación de particiones con PostgreSQL. Las Figuras 3.54, 3.56 y 3.58 muestran un ejemplo de definición de tabla de la BD TPC-E acorde a cada tipo de particionamiento.

```
CREATE TABLE SETTLEMENT (
  SE_T_ID bigint NOT NULL,
  SE_CASH_TYPE varchar(40) NOT NULL,
  SE_CASH_DUE_DATE date NOT NULL,
  SE_AMT decimal(10,2) NOT NULL
) PARTITION BY RANGE (SE_T_ID);
```

Figura 3.54. Ejemplo de definición de tabla con particionamiento Range en la BD TPC-E con PostgreSQL

Posteriormente, se ejecutó el *script* que crea las tablas para cada BD particionada, como se observa en la Figura 3.55, 3.57 y 3.59.

```
tpce_range=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/particiones/range.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.55. Creación de tablas en la BD TPC-E con particionamiento Range en PostgreSQL

```
CREATE TABLE HOLDING_HISTORY (
  HH_H_T_ID bigint NOT NULL,
  HH_T_ID bigint NOT NULL,
  HH_BEFORE_QTY bigint NOT NULL,
  HH_AFTER_QTY bigint NOT NULL
) PARTITION BY LIST (HH_AFTER_QTY);
```

Figura 3.56. Ejemplo de definición de tabla con particionamiento List en la BD TPC-E con PostgreSQL

```
tpce_list=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/particiones/list.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.57. Creación de tablas en la BD TPC-E con particionamiento List en PostgreSQL

```
CREATE TABLE TRADE_HISTORY (
  TH_T_ID bigint NOT NULL,
  TH_DTS timestamp NOT NULL,
  TH_ST_ID varchar(4) NOT NULL
) PARTITION BY HASH (TH_T_ID);
```

Figura 3.58. Ejemplo de definición de tabla con particionamiento Hash con TPC-E en PostgreSQL

```
tpce_hash=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/particiones/hash.ddl
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.59. Creación de tablas en la BD TPC-E con particionamiento Hash en PostgreSQL

3.4.2.8. Fragmentación de la BD TPC-E

Teniendo las bases de datos creadas, se pasó al desarrollo de cada tipo de particionamiento que admite PostgreSQL y se generó un *script* para cada tipo de particionamiento respectivamente. Seguido de ello, se llenó cada BD particionada y por último se agregaron las relaciones, debido a los motivos ya explicados anteriormente. Además, tampoco fue necesario eliminar las restricciones de llave foránea con PostgreSQL en el caso del particionamiento Range y Hash.

3.4.2.9. Particionamiento Range en la BD TPC-E

Al igual que con MySQL, se particionaron solo las tablas TRADE, SETTLEMENT, TRADE_HISTORY y HOLDING_HISTORY, todas particionadas a través de su llave primaria. La Figura 3.60 muestra un ejemplo de particionamiento Range en la tabla TRADE de la BD TPC-E.

```
-- 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);
```

Figura 3.60. Ejemplo de particionamiento Range en la tabla TRADE de TPC-E con PostgreSQL

Además, en la Figura 3.61 se muestra la ejecución del *script* que genera el particionamiento Range en la BD TPC-E.

```
tpce_range=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/particiones/range.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.61. Ejecución del particionamiento Range en la BD TPC-E en PostgreSQL

Posteriormente, se cargaron los datos a las tablas de la BD tpce_range con el mismo *script* utilizado para llenar la BD TPC-E original, tal como se muestra en la Figura 3.62, quedando los datos de las particiones distribuidas de la siguiente manera:

- TRADE: se generaron 10 particiones con 172,800 tuplas cada una.
- SETTLEMENT: se generaron 10 particiones con 172,800 tuplas cada una.
- TRADE_HISTORY: se generaron 10 particiones con una cantidad de entre 414,588 y 415,073 tuplas cada una.
- HOLDING_HISTORY: se generaron 10 particiones con una cantidad de entre 125,631 y 338,472 tuplas cada una.

```
tpce_range=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/load.sql
COPY 7128
COPY 1584
COPY 10
COPY 1598458
COPY 15
COPY 240
COPY 500
COPY 1500
```

Figura 3.62. Llenado de las tablas en la BD TPC-E con particionamiento Range en PostgreSQL

Posteriormente se agregaron las relaciones a la BD tpce_range fragmentada, tal como se observa en la Figura 3.63.

```
tpce_range=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/dss.ri
ALTER TABLE
ALTER TABLE
CREATE INDEX
ALTER TABLE
CREATE INDEX
ALTER TABLE
ALTER TABLE
ALTER TABLE
CREATE INDEX
```

Figura 3.63. Creación de relaciones en la BD TPC-E con particionamiento Range en PostgreSQL

3.4.2.10. Particionamiento List en la BD TPC-E

Para este caso, las tablas particionadas fueron TRADE_HISTORY y HOLDING_HISTORY al igual que en MySQL, la primera se particionó a través de su atributo TH_DTS, logrando así generar particiones por días del mes, la segunda

se particionó a través de su atributo HH_AFTER_QTY, el cual contenía valores que iban desde -800 a 800 en intervalos de 100. La Figura 3.64 muestra un ejemplo de particionamiento List en la tabla HOLDING_HISTORY de la BD TPC-E.

```
-- 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);
```

Figura 3.64. Ejemplo de particionamiento List en la tabla HOLDING_HISTORY de TPC-E en PostgreSQL

Por su parte, en la Figura 3.65 se muestra la ejecución del *script* que genera el particionamiento List en la BD TPC-E.

```
tpce_list=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/particiones/list.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.65. Ejecución del particionamiento List en la BD TPC-E en PostgreSQL

Posteriormente, se cargaron los datos a las tablas de la BD tpce_list, tal como se muestra en la Figura 3.66, quedando los datos de las particiones distribuidos de la siguiente manera:

- TRADE_HISTORY: se generaron 10 particiones con una cantidad de entre 138,215 y 691,288 tuplas cada una.
- HOLDING_HISTORY: se generaron 6 particiones con una cantidad de entre 140,193 y 1,095,746 tuplas cada una.

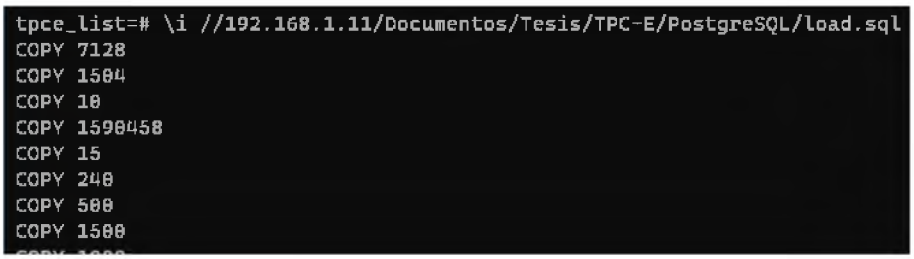


Figura 3.66. Llenado de las tablas en la BD TPC-E con particionamiento List en PostgreSQL

Además, para el particionamiento List, sí fue necesario quitar las llaves primarias y foráneas, tal como se hizo en el particionamiento List con TPC-H, por lo que estas llaves se descartaron en el *script* que se realizó para añadir las relaciones y posteriormente se ejecutó en la BD tpce_list fragmentada, tal como se observa en la Figura 3.67.

Figura 3.67. Creación de relaciones en la BD TPC-E con particionamiento List en PostgreSQL

3.4.2.11. Particionamiento Hash en la BD TPC-E

Para este caso, se particionaron cuatro tablas al igual que en el particionamiento Range. Las tablas particionadas fueron TRADE, SETTLEMENT, TRADE_HISTORY y HOLDING_HISTORY, todas particionadas a través de su llave primaria. La Figura 3.68 muestra un ejemplo de particionamiento Hash en la tabla TRADE_HISTORY de la BD TPC-E.

```
-- 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);
```

Figura 3.68. Ejemplo de particionamiento Hash en la tabla TRADE_HISTORY de TPC-E en PostgreSQL

Además, en la Figura 3.69 se muestra la ejecución del *script* que genera el particionamiento Hash en la BD TPC-E.

```
tpce_hash=# \i //192.168.1.11/Documentos/Tesis/TPC-E/PostgreSQL/particiones/hash.sql
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Figura 3.69. Ejecución del particionamiento Hash en la BD TPC-E en PostgreSQL

Posteriormente, se cargaron los datos a las tablas de la BD tpce_hash, tal como se muestra en la Figura 3.70, quedando los datos de las particiones distribuidos de la siguiente manera:

- TRADE: se generaron 10 particiones con una cantidad de entre 171,957 y 173,741 tuplas cada una.
- SETTLEMENT: se generaron 10 particiones con una cantidad de entre 171,957 y 173,741 tuplas cada una.
- TRADE_HISTORY: se generaron 10 particiones con una cantidad de entre 412,860 y 417,259 tuplas cada una.
- HOLDING_HISTORY: se generaron 10 particiones con una cantidad de entre 223,047 y 226,656 tuplas cada una.

Figura 3.70. Llenado de las tablas en la BD TPC-E con particionamiento Hash en PostgreSQL

Del mismo modo, fue necesario obtener el dato de la cantidad de tuplas para cada partición con el método count(), debido a la forma en que PostgreSQL ejecuta el particionamiento Hash.

Posteriormente, se agregaron las relaciones a la BD tpce_hash fragmentada, tal como se observa en la Figura 3.71.

Figura 3.71. Creación de relaciones en la BD TPC-E con particionamiento Hash en PostgreSQL

3.4.3. MongoDB

Continuando con MongoDB, uno de los aspectos que se tomó en cuenta, es que al ser un SGBD NoSQL, la sintaxis para crear las bases de datos y las consultas cambia drásticamente en comparación al lenguaje SQL utilizado en los SGBD relacionales.

3.4.3.1. Instalación y configuraciones previas

En primera instancia, se descargó MongoDB en su versión 5.0 directamente de su página oficial (*MongoDB*[1], 2022). Esta versión es la más actual a la fecha en que se realizó esta implementación y en este caso se adquirió la versión Enterprise, ya que no requería de pago alguno para el caso de Windows. Al igual que el paquete

de MySQL, este también se encuentra dotado de una herramienta llamada MongoDB Compass, que es una herramienta visual para la conexión con MongoDB.

Posteriormente, se instaló en el equipo donde se ejecutaron las pruebas y se realizaron las siguientes configuraciones previas, para poder realizar el llenado de la BD TPC-E de forma correcta.

- Primeramente, se creó un directorio llamado "data" en la raíz del sistema y dentro de él otro directorio llamado "db". Estos directorios son indispensables para poder ejecutar el servidor (mongod) de MongoDB.
- Además, también se agregó a las variables de entorno de Windows, específicamente a la variable *path*, la ruta donde se instaló MongoDB, que por lo general está ubicado en "C:\Program Files\MongoDB\Server\5.0\bin", esto para poder ejecutar los comandos de MongoDB desde terminal.
- Por último, se descargó e instaló por separado el conjunto de herramientas de BD de MongoDB (*MongoDB*[2], 2022), el cual incluye un conjunto de utilidades de línea de comandos para realizar copias de seguridad de BD y restaurarlas (mongodump y mongorestore) y comandos para exportar e importar colecciones (mongoexport y mongoimport). Además, también fue necesario agregar la ruta de instalación a las variables de entorno.

3.4.3.2. Generación y llenado de la BD TPC-E

Para realizar este proceso, primeramente, se modificaron los archivos que contienen las tuplas para cada tabla de la BD, ya que estos no contaban con los encabezados de las columnas, además de que el caracter por el cual se encontraban separados los atributos era el *pipe* "|", caracter que no es válido en MongoDB para realizar la importación de datos desde archivos CSV. Por tal motivo, se procedió a realizar las modificaciones pertinentes, agregando los encabezados de las tablas, y sustituyendo el caracter separador *pipe* por comas con ayuda del bloc de notas de Windows.

Es importante destacar que, para este caso solo se modificaron los archivos para las tablas TRADE, SETTLEMENT, TRADE_HISTORY y HOLDING_HISTORY, necesarios para implementar los métodos de partición.

Posteriormente, con la ayuda de la herramienta mongoimport, se importaron los archivos CSV a las colecciones de la BD TPC-E. Los SGBD NoSQL tienen la característica de ser no relacionales, por lo que en este caso las relaciones de la BD no se tomaron en cuenta y, por lo tanto, no se agregaron a la BD TPC-E.

La Figura 3.72 muestra un ejemplo del comando utilizado para importar las tablas a la BD TPC-E. En él se especifica el nombre de la BD (-d), seguido del nombre de la colección (-c), el tipo de archivo (--type) e indicar con la opción "--headerline" que el archivo contiene encabezados, ya que, en caso contrario, MongoDB leerá la primer línea como un documento más, y por último el nombre del archivo (--file).

```
D:\Documentos\Tesis\TPC-E\MongoDB\bd_json>mongoimport -d tpce -c trade --type csv --headerlin
e --file "Trade.txt"
2022-10-25T17:49:14.957-0500    connected to: mongodb://localhost/
2022-10-25T17:49:17.958-0500    [.....]  tpce.trade   4.34MB/193MB (2.2%)
2022-10-25T17:49:20.965-0500    [#....]  tpce.trade   9.04MB/193MB (4.7%)
2022-10-25T17:49:23.958-0500    [#....]  tpce.trade   13.0MB/193MB (6.7%)
2022-10-25T17:49:26.961-0500    [##...]  tpce.trade   17.6MB/193MB (9.1%)
2022-10-25T17:49:29.970-0500    [##...]  tpce.trade   22.2MB/193MB (11.5%)
```

Figura 3.72. Importación de la tabla TRADE a la BD TPC-E en MongoDB

Como se observa en la Figura 3.72, no fue necesario indicar el host al cual se conectaría mongoimport, ya que por defecto se conecta a la instancia predeterminada de mongod (localhost:27017). Además, nótese que tampoco fue necesario crear la BD y las colecciones como se hace en los SGBD relacionales; esto es debido a que en MongoDB las BD y las colecciones se crean de forma automática una vez que se empiezan a añadir datos a las colecciones. Además, MongoDB también asigna de forma automática el tipo de dato a los atributos de la colección.

Teniendo la BD original creada, se pasó a realizar una pequeña modificación a la colección TRADE_HISTORY, ya que MongoDB leyó el atributo TH_DTS como cadena (String) y fue necesario modificarla para poder realizar la consulta número ocho de TPC-E, que es una consulta de lectura con filtro por fecha. La Figura 3.73

muestra el comando ejecutado para realizar la modificación de cadena a fecha al atributo TH_DTS de la colección TRADE_HISTORY.

```
MongoDB Enterprise > use tpce
switched to db tpce
MongoDB Enterprise > db.trade_history.updateMany( {},[{$set: {TH_DTS: {$toDate: "$TH_DTS"}}}] )
{
        "acknowledged" : true,
        "matchedCount" : 4148112,
        "modifiedCount" : 4148112
}
```

Figura 3.73. Modificación del atributo TH_DTS a la colección TRADE_HISTORY de TPC-E en MongoDB

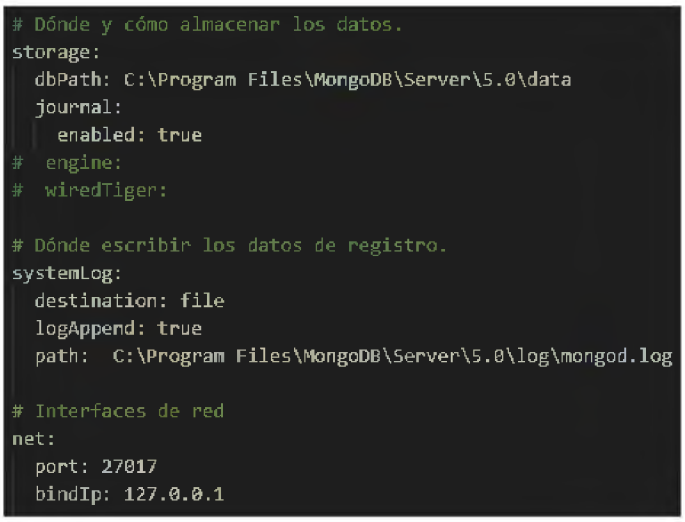
Teniendo lista la BD con todas las colecciones, se pasó a realizar una copia de la BD haciendo uso del comando "mongodump -d tpce", esto con el fin de poder generar la BD para el particionamiento Range y Hash de forma más efectiva.

3.4.3.3. Clúster fragmentado en MongoDB

En MongoDB la fragmentación se realiza sobre lo que se conoce como un clúster fragmentado (comúnmente conocido en inglés como *Sharded Cluster*). Es un proceso totalmente diferente a lo desarrollado con MySQL y PostgreSQL, por lo que fue necesario realizar el curso "*Basic Cluster Administration*" en la academia de MongoDB (*MongoDB*[3], 2022). A continuación, se describe el proceso desarrollado para la configuración de un clúster fragmentado con diez nodos, necesarios para la fragmentación de la BD TPC-E por el método Range y Hash de MongoDB.

A continuación, se describen algunos conceptos de MongoDB, necesarios para la configuración de un clúster fragmentado.

Mongod: el demonio Mongod (del inglés *Daemon*) es el proceso principal de MongoDB que actúa como servidor central de BD. Este proceso contiene todas las opciones de configuración que se utilizan para hacer que la BD sea segura, distribuida y consistente. La Figura 3.74 muestra el archivo de configuración por defecto de Mongod.



```
# Dónde y cómo almacenar los datos.
storage:
  dbPath: C:\Program Files\MongoDB\Server\5.0\data
  journal:
    enabled: true
#  engine:
#  wiredTiger:

# Dónde escribir los datos de registro.
systemLog:
  destination: file
  logAppend: true
  path:  C:\Program Files\MongoDB\Server\5.0\log\mongod.log

# Interfaces de red
net:
  port: 27017
  bindIp: 127.0.0.1
```

Figura 3.74. Archivo de configuración por defecto de Mongod

Mongo: es la aplicación cliente para interactuar con implementaciones MongoDB. Aunque este Shell heredado aún es compatible en MongoDB 5.0, este es ahora obsoleto pasando a ser suplantado por el nuevo MongoDB Shell, el cual se instala de forma independiente y ofrece nuevas ventajas con respecto al heredado Mongo.

Replicación: la replicación es el concepto de mantener múltiples copias de sus datos en distintos nodos (servidores) con el objetivo de asegurar el acceso a los datos, este concepto se conoce como disponibilidad. En MongoDB, un conjunto de réplicas tiene que contar con un mínimo de tres miembros, un primario que es el que recibe las consultas y dos secundarios, esto para garantizar la disponibilidad de los datos ante la posible pérdida de alguno de los nodos.

Fragmentación: en MongoDB, el escalamiento de BD se realiza de forma horizontal, agregando más máquinas y luego distribuyendo el conjunto de datos. La forma en que MongoDB distribuye los datos se denomina *Sharding* (fragmentación de datos en MongoDB), esto permite hacer crecer un conjunto de datos sin la preocupación de almacenar todo en un servidor. En su lugar, MongoDB divide el conjunto de datos en partes y luego distribuye las piezas en tantos fragmentos como se requiera y todo esto junto forma un clúster fragmentado. Además, para garantizar

una alta disponibilidad del clúster fragmentado, cada fragmento se implementa como un conjunto de réplicas.

Mongos: en un clúster fragmentado, las consultas en ocasiones se vuelven un poco complicadas, ya que, si se consulta la BD buscando un documento específico, no resulta obvio dónde buscarlo entre el conjunto de fragmentos. Entonces, entre un clúster fragmentado y los clientes, se configura un proceso denominado "mongos" que actúa como enrutador de consultas. Mongos acepta consultas de clientes y luego determina qué fragmento requiere recibir esa consulta. Por ende, en un clúster fragmentado, los clientes se conectan a Mongos en lugar de conectarse a cada fragmento de forma individual.

Servidor de configuración: Mongos utiliza los metadatos sobre qué datos están contenidos en cada fragmento y estos metadatos se almacenan en los servidores de configuración del clúster fragmentado. Estos datos son muy utilizados por el proceso Mongos, por lo que es necesario asegurar la alta disponibilidad mediante replicación. En un clúster fragmentado se requiere replicar estos datos en los servidores de configuración, entonces, en lugar de un solo servidor de configuración, se implementa un conjunto de réplicas del servidor de configuración.

3.4.3.4. Configuración del clúster fragmentado para la BD TPC-E en MongoDB

Aunque la documentación de MongoDB recomienda un mínimo de tres miembros por cada conjunto de réplicas, para este caso se creó una configuración de diez fragmentos con un solo nodo por cada fragmento, logrando así una distribución de los datos entre los fragmentos, pero sin realizar replicación de datos. Así mismo, solo se creó un servidor de configuración, ya que solo se tiene un nodo por cada fragmento. Está configuración se realizó así debido a los escasos 8 GB de memoria RAM del equipo en donde se desarrollaron las pruebas. Cada proceso Mongod llega a ocupar cerca de dos GB en RAM dependiendo de la carga de trabajo, por lo que si se implementaban tres nodos por cada fragmento, se necesitan cerca de 60 GB de memoria RAM para poder ejecutar las pruebas. Este

comportamiento es lógico, dado que este tipo de configuración está hecho para realizarse sobre clústeres o en un conjunto de equipos de cómputo.

Un proceso/instancia Mongod o Mongos, se configura por opciones de línea de comandos o con archivos de configuración en formato YAML (lenguaje de serialización de datos fácil de usar para todos los lenguajes de programación). Para este caso se utilizaron archivos de configuración para la configuración de cada proceso/instancia Mongod o Mongos.

A continuación, se describen los pasos realizados para la configuración de un clúster fragmentado con 10 nodos (fragmentos) para la BD TPC-E, que es lo equivalente a lo realizado en MySQL y PostgreSQL con los métodos de fragmentación Range y Hash.

1. **Crear los archivos de configuración para cada fragmento**. Para este caso, se realizó la configuración mínima necesaria para cada nodo y solo se añadieron los apartados para el almacenamiento de los datos (es necesario que el directorio esté creado) y las interfaces de red para la conexión, así como el apartado *sharding:* en donde se indica que ese nodo es candidato para ser usado como fragmento con la opción "clusterRole: shardsvr", y *replication*: en donde se asigna un nombre al conjunto de réplicas. Para este caso los fragmentos se nombraron como shard1 a shard10. La Figura 3.75 muestra un ejemplo del archivo de configuración para el primer fragmento (shard1). Para hacer posible la ejecución en la misma máquina, se cambió el puerto de escucha para cada fragmento y van desde el puerto 27001 al 27010.

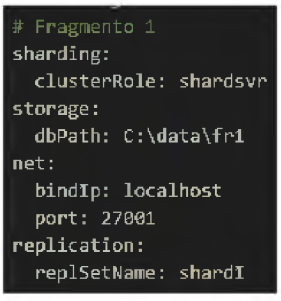


Figura 3.75. Archivo de configuración para el primer fragmento del clúster

2. **Ejecución de cada fragmento**. A continuación, se ejecutó un proceso Mongod por cada fragmento con su respectivo archivo de configuración indicando el nombre del archivo con la opción "-f". La Figura 3.76 muestra un ejemplo de ejecución del primer fragmento.

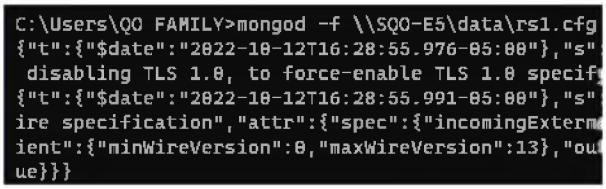


Figura 3.76. Ejecución del primer fragmento

Enseguida, se realizó la conexión a ese fragmento desde otra terminal con el cliente Mongo utilizando la opción "--port" para conectarse a través del puerto, o la opción "--host" para conectarse a través de la IP/HostName del equipo seguido del puerto, y se inicializa con el comando "rs.initiate()". Si desea agregar más nodos a ese fragmento para que actúen como réplicas, se realiza con el comando "rs.add()". La Figura 3.77 muestra un ejemplo de conexión e inicialización para el primer fragmento. Además, con el comando "rs.isMaster()", es posible verificar si el nodo actual es el primario y para terminar la conexión se usa el comando "exit".

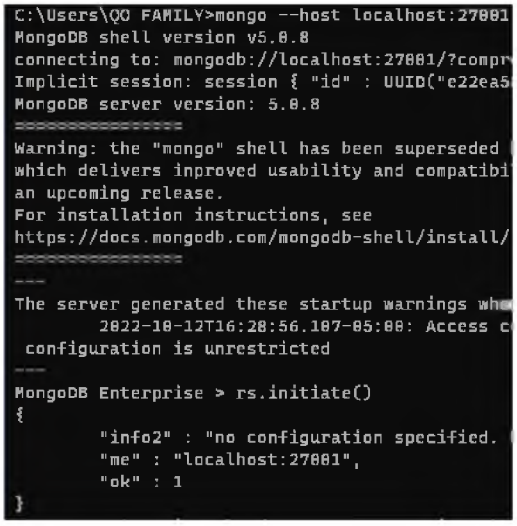


Figura 3.77. Conexión e inicialización del primer fragmento

3. **Crear archivo de configuración para el servidor de configuración**. Este archivo de configuración es muy similar al de los fragmentos, solo se

modificó el rol para ese nodo en el apartado *sharding:* con la opción "clusterRole: configsvr", indicando que ese nodo se usa como servidor de configuración. Además, se asignó el puerto de escucha 26001 y se colocó el nombre "csrs" al conjunto de réplicas, ya que, como se mencionó anteriormente, el servidor de configuración también requiere ejecutarse como un conjunto de réplicas al igual que los fragmentos. En este caso sólo se requirió de un servidor de configuración, ya que por cada fragmento sólo se tiene un nodo. La Figura 3.78 muestra el archivo de configuración para el servidor de configuración.

```
# servidor de configuración
sharding:
  clusterRole: configsvr
storage:
  dbPath: C:\data\csrs
net:
  bindIp: localhost
  port: 26001
replication:
  replSetName: csrs
```

Figura 3.78. Archivo de configuración para el servidor de configuración

4. **Ejecución del servidor de configuración**. La ejecución se realizó exactamente igual que los fragmentos mencionados en el punto dos. La Figura 3.79 muestra la ejecución del servidor de configuración.

```
C:\Users\QO FAMILY>mongod -f \\SQO-E5\data\csrs.cfg
{"t":{"$date":"2022-10-12T16:42:08.306-05:00"},"s":"
 specification","attr":{"spec":{"incomingExternalCli
t":{"minWireVersion":0,"maxWireVersion":13},"outgoin
}}
```

Figura 3.79. Ejecución del servidor de configuración

En seguida, se conecta a ese nodo que actúa como servidor de configuración y se inicializa al igual que los fragmentos. La Figura 3.79 muestra la conexión e inicialización del servidor de configuración.

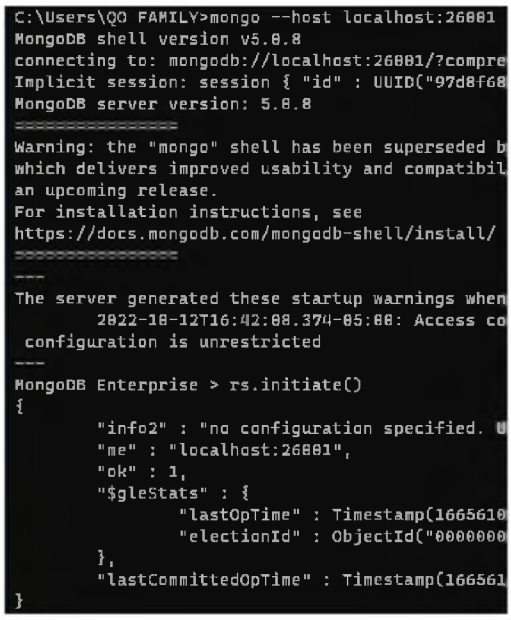


Figura 3.80. Conexión e inicialización del archivo de configuración

5. **Crear archivo de configuración de Mongos**. Como ya se mencionó anteriormente, Mongos es un intermediario entre el cliente y el clúster fragmentado, el cuál lee constantemente los metadatos que contiene el servidor de configuración. Por tal motivo, Mongos no requiere de un directorio donde almacenar datos, solo requiere las interfaces de red para la conexión y el servidor de configuración. Este se agrega en el apartado *sharding:* indicando en la opción "configDB" lo siguiente; "replSetName/IP:puerto": replSetName es el nombre del servidor de configuración; IP es la dirección IP o HostName de la computadora, y el puerto. Si existieran más nodos para la replicación del servidor de configuración, estos se agregan de la siguiente manera: "replSetName/IP_1:puerto_1,...,IP_n:puerto_n". La Figura 3.81 muestra el archivo de configuración del servidor de configuración, el cual se ejecuta en el puerto 26000.

```
# archivo de configuración de mongos
sharding:
  configDB: csrs/localhost:26001
net:
  bindIp: localhost
  port: 26000
```

Figura 3.81. Archivo de configuración de Mongos

6. **Ejecución del proceso Mongos.** Un proceso Mongos se ejecuta de forma similar a un proceso Mongod. La Figura 3.82 muestra la ejecución del proceso Mongos.

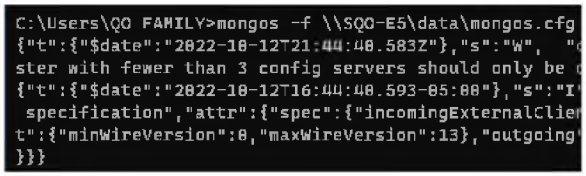


Figura 3.82. Ejecución del proceso Mongos

En seguida, se conecta a Mongos con “mongo --port 26000”, y se agrega cada fragmento en el clúster. La Figura 3.83 muestra un ejemplo de cómo agregar un fragmento al clúster con el comando “sh.addShard()”, el cual recibe como parámetro el nombre del fragmento y el host de forma similar al servidor de configuración que se agregó en el archivo de configuración de Mongos.

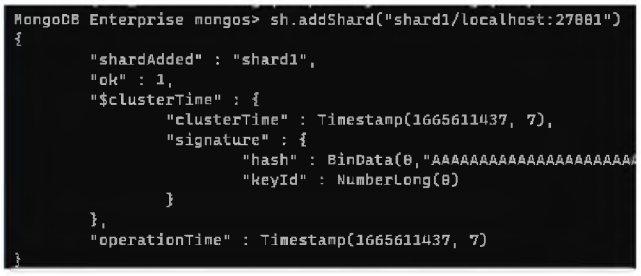


Figura 3.83. Adición del primer fragmento al clúster

Además, con el comando “sh.status()”, se verifica el estado actual del clúster fragmentado. Este comando muestra información sobre los fragmentos del clúster (apartado *shards*), las BD y cómo estas se encuentran distribuidas en los fragmentos, entre otros aspectos. La Figura 3.84 muestra los 10 fragmentos añadidos al clúster haciendo uso del comando “sh.status()”.

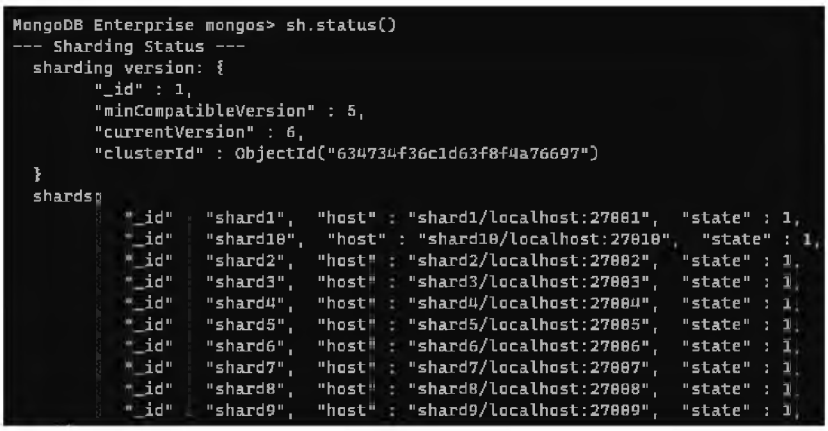


Figura 3.84. Configuración del clúster fragmentado con 10 fragmentos

3.4.3.5. Fragmentación de la BD TPC-E

Hasta este punto, ya se tiene un clúster fragmentado con 10 fragmentos, pero aún no se crean las BD. En MongoDB, los datos de una colección se distribuyen de forma autónoma en un clúster fragmentado, pero hay que indicarle en qué BD se requiere habilitar la fragmentación y sobre qué colecciones se requiere efectuar la distribución de los datos. A continuación, se describen los pasos seguidos para habilitar la fragmentación para las BD que se fragmentaron con el método Range y Hash.

El primer paso para que se efectúe la fragmentación en una BD, es habilitarlo por medio del comando “sh.enableSharding()”, el cual recibe como parámetro el nombre de la BD a fragmentar. Como se mencionó anteriormente, en MongoDB las BD se crean de forma automática, por lo que con el comando anterior, esta BD se crea y queda almacenada hasta que se inserten datos en ella. La Figura 3.85 muestra un ejemplo de la ejecución del comando para habilitar la fragmentación en la BD nombrada tpce_range y de igual forma se realizó para otra BD nombrada tpce_hash.

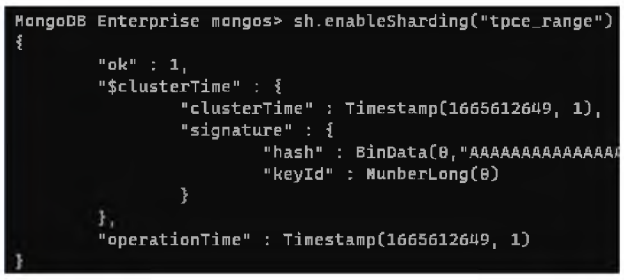


Figura 3.85. Habilitando fragmentación en la BD TPC-E para el particionamiento Range

En MongoDB, para lograr una distribución de los datos de las colecciones, es necesario crear un nuevo índice que funge como clave de fragmento, y es a través de esa clave que Mongos sabe cómo distribuir los datos en los fragmentos y lo hace de forma automática. La Figura 3.86 muestra la ejecución del comando "db.colección.createIndex()" para la colección TRADE de la BD tpce_range. Este comando recibe como parámetro el nombre del atributo que funge como clave de fragmento y un "1" para indicar que es una clave Range o "hashed" para indicar que es una clave Hash. Además, este mismo comando se ejecutó para todas las colecciones que se fragmentaron en MySQL y PostgreSQL con los mismos atributos utilizados en la partición Range y Hash.

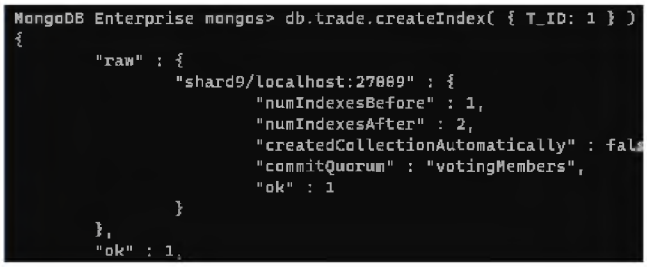


Figura 3.86. Creación de la clave de fragmento Range en la colección TRADE de TPC-E

Posteriormente, se ejecutó el comando "sh.shardCollection()", el cual recibe como parámetro el nombre de la BD junto con la colección a fragmentar, seguido de la clave de fragmento, tal como se especificó en el comando "createIndex()". Este comando es el que da inicio al proceso de distribución de los datos de la colección en los 10 fragmentos del clúster. La Figura 3.87 muestra un ejemplo de ejecución del comando "sh.shardCollection()" en la colección TRADE de la BD tpce_range, y esto mismo se ejecutó para cada colección de la BD tpce_range y tpce_hash.

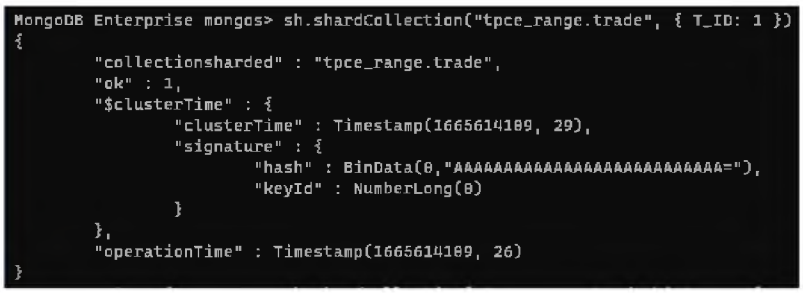


Figura 3.87. Fragmentación de la colección TRADE con clave de fragmento Range en la BD TPC-E

Finalmente, solo queda cargar los datos de la BD TPC-E haciendo uso del comando mongorestore, de la BD generada anteriormente con mongodump. Al comando mongorestore se le especifica el puerto donde se ejecuta el proceso Mongos con la opción "--port", ó "-h" para indicar el host; el nombre de la BD a la cual se cargan los datos con la opción "-d" y al final la ruta del directorio creado con mongodump. La Figura 3.88 muestra un ejemplo de cómo importar la BD tpce_range con mongorestore y de forma similar se realizó con la BD tpce_hash.

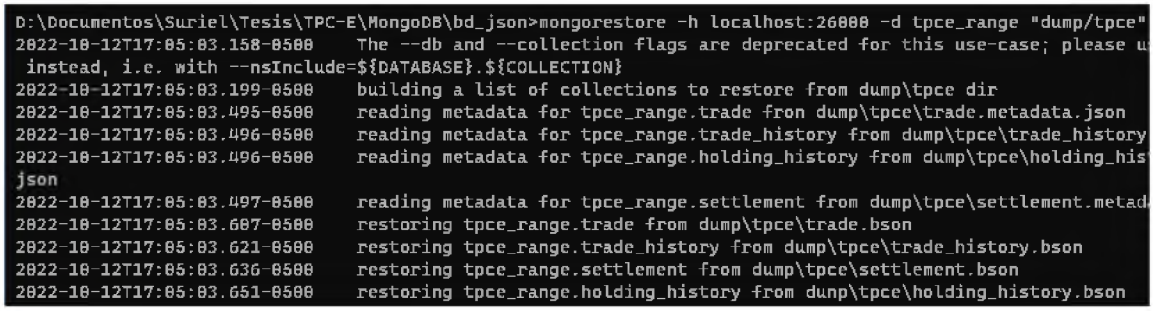


Figura 3.88. Restauración de la BD TPC-E original a la BD con particionamiento Range

Con "sh.status()" se verifica el estado del clúster fragmentado y aunque en ocasiones parezca que los datos no se están distribuyendo, esto es porque, cuando Mongos no recibe ninguna operación, el balanceador que es el encargado de distribuir los datos, no se ejecutará. Para solucionar esto, es necesario insertar un documento a cualquiera de las colecciones o ejecutando la consulta uno de TPC-E, por ejemplo. Luego de unos minutos se verifica con el comando "count()" la cantidad de documentos en cada colección, cuando el balanceador termine de distribuir los datos, el comando arroja la cantidad total de documentos de la colección, en caso contrario muestra un número inexacto. La Figura 3.89 y 3.90 muestran cómo se

encuentran distribuidos los datos en el clúster fragmentado de la BD tpce_range y tpce_hash.

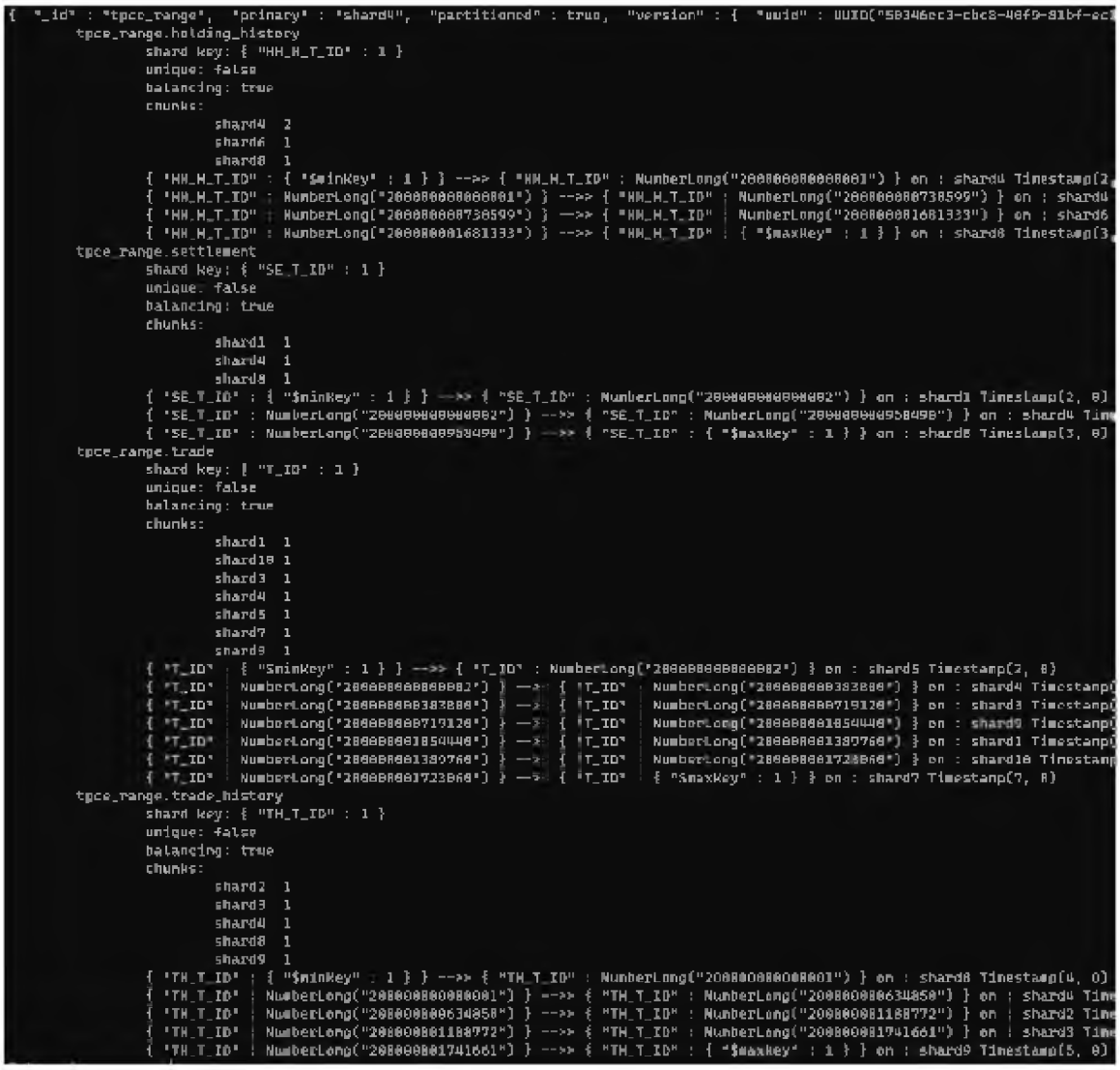


Figura 3.89. Distribución de los datos de las colecciones en la BD tpce_range

Como se observa en la Figura 3.89 y 3.90, en el apartado “chunks” es donde se describe en qué fragmentos se encuentran distribuidos los datos de cada colección. Esto se verifica conectándose directamente a algunos de los fragmentos y verificando la cantidad de documentos que tiene cada colección por medio del comando count(), no obstante es importante mencionar que las conexiones reales siempre son a través del proceso Mongos.

Figura 3.90. Distribución de los datos de las colecciones en la BD tpce_hash

3.5. Etapa de comparación

3.5.1. Operaciones en la BD TPC-H original y fragmentada

Una vez creadas las bases de datos necesarias para TPC-H, se pasó a desarrollar un código en Java con NetBeans IDE en su versión 12, el cual se conecta a los distintos SGBD con el fin de automatizar la ejecución de las consultas que el propio *benchmark* proporciona. Los códigos desarrollados para la ejecución de las pruebas con TPC-H se presentan en el anexo A.

El *benchmark* TPC-H consta de 22 consultas de lectura de datos, cada consulta se ejecutó 5 veces para obtener un tiempo promedio de ejecución para cada consulta. La Figura 3.91 muestra a modo de ejemplo la consulta número uno de TPC-H. Al término de la ejecución del programa, este arroja un archivo de texto plano con el tiempo promedio de respuesta (en formato de segundos) para cada consulta ejecutada tal como se observa en la Figura 3.92, de acuerdo con el SGBD y la BD indicados en el código. Además, en el caso de MongoDB se omitieron las pruebas con TPC-H debido a la complejidad de las consultas y el tiempo excesivo para implementar las consultas al lenguaje que estos SGBD NoSQL utilizan.

Por otra parte, es importante mencionar que la consulta número quince de TPC-H realiza la creación de una vista (tablas virtuales que muestran datos almacenados de otras tablas), por lo que fue necesario crear este *view* para cada BD TPC-H en los distintos SGBD relacionales. El código realizado lee las consultas desde archivos de texto plano y al hacer esto se logró ejecutar dicha consulta sin ningún problema.

Otro aspecto importante por destacar es, que en PostgreSQL las consultas número 17 y 20 tardaron mucho tiempo en ejecutarse en la BD original de TPC-H, por tal motivo, solo se ejecutó una vez cada consulta en cada una de las BD fragmentadas.

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
        l_shipdate <= date '1998-12-01' - interval '90' day
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus;
```

Figura 3.91. Consulta número uno en la BD TPC-H

```
QUERY 1
Tiempo promedio de respuesta: 17.172642125000003

QUERY 2
Tiempo promedio de respuesta: 0.402785925

QUERY 3
Tiempo promedio de respuesta: 82.11534515000001

QUERY 4
Tiempo promedio de respuesta: 4.47474495

QUERY 5
Tiempo promedio de respuesta: 33.007400725000004

QUERY 6
Tiempo promedio de respuesta: 3.997598275
```

Figura 3.92. Archivo de salida con los tiempos de ejecución de la BD TPC-H original

3.5.2. Operaciones en la BD TPC-E original y fragmentada

Una vez creadas las bases de datos necesarias para TPC-E, se pasó a desarrollar al igual que para TPC-H, un código en Java el cual se conecta a los distintos SGBD con el fin de automatizar la ejecución de las consultas que se generaron, ya que en este caso no se utilizaron las proporcionadas por el *benchmark*. Los códigos desarrollados para la ejecución de las pruebas con TPC-E se presentan en el anexo A.

Se generaron en total 8 consultas (Tabla 3.4 y 3.5), dos por cada tipo de operación en la BD (creación, lectura, actualización y eliminación) y cada consulta se ejecutó 100 veces para obtener un tiempo promedio de ejecución por cada consulta. En este caso se realizó un mayor número de ejecuciones, ya que no tardaban tanto tiempo a diferencia de TPC-H donde algunas consultas sí tardaron varios minutos, incluso horas en ejecutarse. Al final el código da como salida un archivo de texto plano con el tiempo promedio de ejecución (en formato de segundos) para cada consulta ejecutada, tal como se observa en la Figura 3.93 de acuerdo con el SGBD y la BD indicados en el código.

Además, en este caso sí fue posible ejecutar las pruebas con TPC-E en todos los SGBD, ya que las consultas elaboradas no son tan complejas como en el caso de TPC-H donde sí se utilizaron las que el *benchmark* proporciona.

Por otro lado, dado que este *benchmark* realiza consultas de escritura de datos, las consultas se diseñaron para poder ejecutar sin problemas las consultas de inserción y eliminación. Para ello, se manipuló con código el identificador 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.

Tabla 3.4. Consultas de lectura y escritura para la BD TPC-E en MySQL y PostgreSQL

No. de consulta	Predicado
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;

Tabla 3.5. Consultas de lectura y escritura para la BD TPC-E en MongoDB

No. de consulta	Predicado

1	db.trade.insertOne(
	{
	T_ID: id,
	T_DTS: "2005-01-11",
	T_ST_ID: "CMPT",
	T_TT_ID: "TLB",
	T_IS_CASH: "1",
	T_S_SYMB: "AGNPRA",
	T_QTY: 800.0,
	T_BID_PRICE: 20.55,
	T_CA_ID: 4.3000002741E10,
	T_EXEC_NAME: "Jose Citino",
	T_TRADE_PRICE: 20.51,
	T_CHRG: 5.0,
	T_COMM: 59.07,
	T_TAX: 1996.5,
	T_LIFO: "0"
	}
	)

2	db.settlement.insertOne(
	{
	SE_T_ID: id,
	SE_CASH_TYPE: "Cash Account",
	SE_CASH_DUE_DATE: "2005-01-18",
	SE_AMT: -2117.75
	}
	)

No. de consulta	Predicado
3	db.trade_history.find({ TH_DTS: { $gte: ISODate("2005-01-03"), $lte: ISODate("2005-01-05") } }, { _id: 0, TH_DTS: 1 })
4	db.trade_history.find({ TH_ST_ID: {$nin: ["SBMT","PNDG"]} })
5	db.holding_history.updateMany({HH_H_T_ID: {$gte: 200000001641638}}, [{ $set: { HH_BEFORE_QTY: "$HH_BEFORE_QTY" } }])
6	db.holding_history.updateMany({HH_H_T_ID: {$lte: 200000000075031}}, [{ $set: { HH_AFTER_QTY: "$HH_AFTER_QTY" } }])
7	db.settlement.deleteOne({ SE_T_ID: id })
8	db.trade.deleteOne({ T_ID: id })

QUERY 1
Tiempo promedio de respuesta: 8.720525000000003E-4

QUERY 2
Tiempo promedio de respuesta: 3.1151999999999997E-4

QUERY 3
Tiempo promedio de respuesta: 2.8661139000000007

QUERY 4
Tiempo promedio de respuesta: 3.077301662499999

QUERY 5
Tiempo promedio de respuesta: 0.09505926000000003

QUERY 6
Tiempo promedio de respuesta: 0.23180198249999986

QUERY 7
Tiempo promedio de respuesta: 5.438199999999999E-4

QUERY 8
Tiempo promedio de respuesta: 6.486374999999998E-4

Figura 3.93. Ejemplo del archivo de salida con los tiempos de ejecución de las consultas en TPC-E

3.6. Etapa de caracterización y discriminación

3.6.1. Descripción de los criterios de caracterización y discriminación

Con base en los experimentos realizados para la creación de la BD TPC-H y TPC-E en los distintos SGBD se encontraron algunos puntos a destacar.

Como primer punto, es importante mencionar que en MySQL se encontró con el problema de que el particionamiento de tablas no se efectúa sobre índices que son llave foránea, por ende, se desarrolló el *script* que elimina las llaves foráneas de las tablas en las que se realizó el particionamiento y aquellas con las que tenían relación y esto se aplicó en cada BD antes de ser particionada, lo que provocó una posible baja en el rendimiento, ya que MySQL depende en gran medida de estas relaciones para mejorar su rendimiento. Otra posible solución es modificar las tablas agregando los índices a través de los cuales se realizó el particionamiento al conjunto de llaves primarias, pasando a ser una llave primaria compuesta. Sin embargo, esto desencadena otras posibles problemáticas con los gestores.

Además, en PostgreSQL fue necesario especificar desde la creación de las tablas el tipo de particionamiento a emplear, ya que no es posible alterar las tablas luego de ser creadas, a diferencia de MySQL en donde sí es posible modificarlas. Por tal motivo, se desarrolló un *script* por cada tipo de particionamiento para la creación de las tablas, definiendo en cada uno de ellos el tipo de particionamiento que se utilizó. No obstante, es importante mencionar que en PostgreSQL es posible añadir una tabla regular o particionada existente como una partición de una tabla particionada por medio del comando *ATTACH PARTITION* (*PostgreSQL*[1], 2021). Por su parte, a diferencia de MySQL, gracias a las pruebas realizadas se observó que PostgreSQL no depende tanto de las relaciones en las tablas, ya que aún sin ellas, es capaz de obtener un buen rendimiento para la mayoría de las consultas, en este caso de TPC-H y TPC-E.

Por otra parte, en PostgreSQL cuando se genera el particionamiento, el SGBD crea las tablas de tal forma que es posible acceder fácilmente a ellas de forma global como si de una tabla normal se tratase, algo que MySQL maneja de

forma diferente ya que es necesario hacer referencia a la tabla principal e indicar la partición a la cual se quiere acceder.

Además, para el particionamiento List con PostgreSQL, fue necesario quitar las llaves primarias y foráneas, tal como se hizo en MySQL con este mismo particionamiento ya que PostgreSQL no permite particionar tablas si la columna utilizada para el particionamiento no se encuentra definida como llave primaria.

Por otro lado, algunos tipos de particionamiento como el Key de MySQL y el Hash de PostgreSQL, distribuyen de forma no equitativa los datos en las particiones, por lo que fue necesario obtener el dato de la cantidad de tuplas con la función count().

Con MongoDB, la implementación fue bastante distinta en comparación con los SGBD relaciones ya que básicamente es configurar un clúster fragmentado en modo de alto rendimiento al implementar un conjunto de réplicas por cada fragmento. Aunque se requiere del estudio de varios conceptos para lograr un buen entendimiento sobre cómo realizar esta configuración, es importante destacar que el hecho de que los datos se distribuyen de forma totalmente automática conforme se van agregando datos a las colecciones, es un punto bastante a favor para MongoDB ya que en los SGBD relacionales no ocurre así, por el contrario es necesario volver a particionar las tablas para lograr un mejor balance de los datos en el caso de MySQL y PostgreSQL.

Capítulo 4. Resultados

4.1. Comparación del tiempo de respuesta de las operaciones en la BD TPC-H original contra la fragmentada

Los resultados del experimento se obtuvieron utilizando una computadora de escritorio con las siguientes características.

- 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

Las Tablas 4.1 y 4.2 muestran los tiempos de ejecución (en formato de segundos) de las consultas de solo lectura en TPC-H.

Tabla 4.1. Comparación de tiempos de ejecución en la BD TPC-H con MySQL

Número de consulta	Sin partición	Con partición Tipo de partición			
		Range	List	Hash	Key
TC1	**17.17**	18.22	21.36	17.82	17.40
TC2	**0.40**	4.56	0.42	7.51	0.53
TC3	82.12	63.26	**40.07**	94.11	85.73
TC4	**4.47**	5.49	16.12	7.60	7.31
TC5	33.01	50.49	**14.62**	84.42	74.04
TC6	4.00	4.37	**1.37**	7.35	4.28
TC7	26.06	**19.90**	22.96	33.60	29.69
TC8	115.90	**93.86**	8,321.72	148.55	132.14
TC9	144.83	270.94	**65.84**	494.56	132.16
TC10	27.86	181.41	**15.22**	274.26	267.42
TC11	7.88	**5.12**	6.91	7.23	7.08
TC12	**5.88**	6.25	10.80	8.68	7.52
TC13	258.38	**160.30**	225.53	266.84	228.90
TC14	12.98	815.57	**5.23**	1,162.47	1,074.17
TC15	8.55	9.16	**2.97**	11.92	11.62
TC16	5.89	**3.42**	5.53	4.83	4.75
TC17	**2.11**	2.28	2.45	2.67	2.65
TC18	**5.52**	162.97	363.99	243.97	235.68

Número de consulta	Sin partición	Con partición Tipo de partición Range	List	Hash	Key
TC19	3.03	**2.71**	3.09	4.02	3.78
TC20	12.89	65.19	**2.33**	83.08	79.23
TC21	**15.92**	17.93	34.28	19.95	19.19
TC22	0.72	**0.68**	2.14	1.01	0.88

Tabla 4.2. Comparación de tiempos de ejecución en la BD TPC-H con PostgreSQL

Número de consulta	Sin partición	Con partición Tipo de partición Range	List	Hash
TC1	**3.3646**	3.4093	3.4648	3.4216
TC2	**0.4671**	0.4750	0.5118	0.4941
TC3	**0.7304**	3.8190	1,281.5469	4.0590
TC4	**0.4190**	2.1048	2.1616	2.0284
TC5	**0.4803**	3.3747	3.5286	3.4435
TC6	0.6128	**0.5421**	0.5913	0.6104
TC7	**0.6800**	4.6929	4.3754	4.2540
TC8	1.0327	2.0101	**0.9496**	2.0292
TC9	**2.2148**	41.5258	2.6066	42.0131
TC10	**0.9750**	2.8005	1.0798	2.3949
TC11	0.2529	0.3025	**0.2362**	0.3063
TC12	0.9003	0.8509	0.8880	**0.8348**
TC13	1.1560	1.0413	1.0800	**1.0310**
TC14	0.6237	0.6258	**0.5627**	0.6176
TC15	1.3041	1.1532	**1.0991**	1.1536
TC16	0.6467	**0.6233**	0.6363	0.6942
TC17	**5,116.8409**	5,256.9270	5,520.4596	5,196.5886
TC18	**3.6895**	9.7228	48.5755	9.7740
TC19	0.8783	0.8469	**0.7626**	0.8391
TC20	30,839.1321	31,456.1954	33,980.4296	**30,745.5285**
TC21	0.9784	0.9624	4.2166	**0.9487**
TC22	0.6812	**0.6347**	0.6697	0.6352

4.2. Comparación del tiempo de respuesta de las operaciones en la BD TPC-E original contra la fragmentada

De forma similar, la Tabla 4.3 muestra los tiempos de ejecución de las consultas de lectura y escritura de datos en TPC-E.

Tabla 4.3. Comparación de tiempos de ejecución en la BD TPC-E

SGBD	Versión	Número de consulta	Tipo de consulta	Sin partición	Con partición Tipo de partición			
					Range	List	Hash	Key
MySQL	8.0	TC1	C	0.00087	0.00103	0.00106	0.00091	**0.00086**
		TC2	C	**0.00031**	0.00046	0.00046	0.00048	0.00046
		TC3	R	**2.86611**	3.03084	5.49348	3.00366	3.07758
		TC4	R	**3.07730**	3.32959	5.86820	3.31341	3.34779
		TC5	U	0.09506	**0.09412**	2.92519	0.10711	0.10437
		TC6	U	0.23180	**0.22921**	3.18076	0.24685	0.24427
		TC7	D	0.00054	0.00040	0.00040	**0.00029**	0.00043
		TC8	D	0.00065	0.00054	0.00046	**0.00044**	0.00051
PostgreSQL	14.1	TC1	C	**0.00111**	0.00121	0.001424	0.002188	-
		TC2	C	**0.00052**	0.00099	0.001077	0.001184	-
		TC3	R	**0.24385**	0.55093	0.338835	0.549119	-
		TC4	R	**1.68750**	1.84401	2.189544	1.850861	-
		TC5	U	0.77639	0.84029	**0.428474**	0.766166	-
		TC6	U	2.76766	2.15018	**1.754820**	2.305706	-
		TC7	D	0.00045	0.00041	**0.000399**	0.000444	-
		TC8	D	0.00076	0.00147	**0.000700**	0.001378	-
MongoDB	5.0	TC1	C	**0.00227**	0.01632	-	0.01704	-
		TC2	C	**0.00086**	0.01566	-	0.01550	-
		TC3	R	**0.00207**	1.18503	-	0.00656	-
		TC4	R	**0.00137**	0.00349	-	0.00590	-
		TC5	U	2.08033	**0.33047**	-	0.67696	-
		TC6	U	2.47448	1.24665	-	**0.79823**	-
		TC7	D	1.37265	**0.01555**	-	0.01568	-
		TC8	D	1.37682	**0.01547**	-	0.01553	-

4.3. Análisis y discusión

Como se observa en la Tabla 4.1, el particionamiento Range logró mejorar los tiempos de respuesta para siete de las consultas TPC-H, siendo las consultas TC7, TC8, TC11, TC13, TC16, TC19 y TC22 aquellas que lograron dicha mejora. Por su parte, el particionamiento List también logró una mejora en el tiempo de ejecución de las consultas TC3, TC5, TC6, TC9, TC10, TC14, TC15 y TC20, 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 4.2), el particionamiento Range logró mejorar los tiempos de respuesta para tres de las consultas de TPC-H, siendo las consultas TC6, TC16 y TC22 aquellas que lograron dicha mejora. Además, el particionamiento List logró una mejora en el tiempo de ejecución de las consultas TC8, TC11, TC14, TC15 y TC19. Así mismo, el particionamiento Hash logró una mejora en el tiempo de ejecución de las consultas TC12, TC13, TC20 y TC21 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.

Posteriormente, en la Tabla 4.3 se observa que, 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 TC5 y TC6 aquellas que lograron dicha mejora. Por su parte, el particionamiento Hash logró una mejora en el tiempo de ejecución de las consultas TC7 y TC8. Además, el particionamiento Key también logró una mejora en el tiempo de ejecución de la consulta uno, obteniendo así una mejora en 5 de las 8 consultas, es decir, se obtuvo una mejora total del 62.5% en las BD fragmentadas con respecto a la BD original.

En cuanto a los tipos de operación que se ejecutaron, se observa que hubo mejora en las operaciones de escritura de datos. Por su parte, el particionamiento Key logró mejorar el tiempo de ejecución de la operación de creación TC1, el particionamiento Range logró mejorar el tiempo de ejecución de las operaciones de actualización (TC5 y TC6) y el particionamiento Hash logró mejorar el tiempo de ejecución de las operaciones de eliminación (TC7 y TC8).

Siguiendo con PostgreSQL (Tabla 4.3), es posible apreciar que el particionamiento List obtuvo una mejora en las consultas TC5, TC6, TC7 y TC8, lo que indica que hubo una mejora de rendimiento total en las BD fragmentadas de sólo un 50%.

Además, tomando en cuenta los tipos de operación ejecutados, el particionamiento List mejoró el tiempo de ejecución de las operaciones de actualización (TC5 y TC6) y eliminación (TC7 y TC8).

Continuando con MongoDB (Tabla 4.3), se observa que el particionamiento Range mejoró los tiempos de respuesta para las consultas TC5, TC7 y TC8 de TPC-E y solo la consulta TC6 se mejoró a través del particionamiento Hash, obteniendo así una mejora en 4 de las 8 consultas, es decir, se obtuvo una mejora total del 50% en las BD fragmentadas con respecto a la BD original.

Con respecto a los tipos de operación ejecutados, se observa que hubo mejora en las operaciones de escritura de datos. Por su parte, el particionamiento Range logró mejorar el tiempo de ejecución de la operación de actualización TC5 y las dos operaciones de eliminación (TC7 y TC8), y solo la operación de actualización TC6 se logró mejorar con el particionamiento Hash.

En la Tabla 4.4, se muestran de forma breve las ventajas y desventajas encontradas en cada gestor con respecto a la partición, así como los tipos de operación que se optimizaron con cada tipo de partición.

Tabla 4.4. Discriminación de los SGBD respecto a la partición de tablas

SGBD	Ventajas	Desventajas	Tipo de partición	Operaciones optimizadas
MySQL	➢ La partición se efectúa sin implementar un conjunto de réplicas. ➢ Se particiona una tabla desde su creación o una existente. ➢ Permite la subpartición de tablas. ➢ Posee métodos de partición adicionales, basados en los fundamentales (los implementados en esta tesis).	➢ El rendimiento se ve afectado por la ausencia de llaves primarias y foráneas. ➢ Si existen llaves foráneas, éstas se eliminan para lograr la partición. ➢ Si la tabla contiene llaves primarias, solo se particiona la tabla por medio de ese atributo.	Range	Lectura y actualización
			List	Lectura
			Hash	Eliminación
			Key	Inserción
PostgreSQL	➢ La partición se efectúa sin implementar un conjunto de réplicas. ➢ El rendimiento no se ve afectado por la ausencia de llaves primarias y foráneas. ➢ Ofrece mayor flexibilidad al utilizar otras formas de partición alternativas, como las vistas.	➢ Es necesario indicar desde la creación de la tabla el tipo partición y no es posible particionar una tabla existente. ➢ Si la tabla contiene llaves primarias, solo se particiona la tabla por medio de ese atributo.	Range	Lectura
			List	Lectura, actualización y eliminación
			Hash	Lectura
MongoDB	➢ Distribución automática de los datos en los fragmentos. ➢ Se agregan más fragmentos por separado.	➢ Requiere implementar un conjunto de réplicas. ➢ Requiere más recursos de hardware para realizarse en una sola computadora.	Range	Actualización, eliminación
			Hash	Actualización

Capítulo 5. Conclusiones

5.1. 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.

Además, con base en la experimentación realizada con las BD TPC-H y TPC-E, el tipo de partición que mejores resultados obtuvo fue la partición List de PostgreSQL, al lograr mejorar los tiempos de ejecución en operaciones de lectura, actualización y eliminación. Por otra parte, se concluye que la BD TPC-H resulta ser la más adecuada para medir el rendimiento de una BD junto con las consultas que el *benchmark* proporciona, ya que se explota todo el potencial que ofrece el SGBD en términos de rendimiento. Además, con las pruebas realizadas se concluye que es complejo convertir las consultas de TPC-H al lenguaje de un SGBD NoSQL.

Por otra parte, en cuanto a los resultados obtenidos se observó que MySQL fue el que logró mejores resultados al obtener una mejora de rendimiento en por lo menos una consulta por cada tipo de operación, no obstante, es necesario recalcar que cada SGBD tiene sus ventajas y desventajas, aunque para el caso de la partición 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 estos. Además, en MySQL también es posible realizar subparticiones, lo que otorga un rendimiento extra si la fragmentación se implementa de forma adecuada. No obstante, si es complejo realizar un método de partición adecuado para la BD, el mejor SGBD es MongoDB, ya que implementa la partición Hash para lograr una distribución uniforme de los datos en los fragmentos de forma automática.

5.2. Trabajo futuro

Finalmente, como trabajo a futuro se pretende implementar la partición en la BD TPC-E con el SGBD NoSQL Redis, para conocer las diferencias que existen con respecto a la fragmentación de MongoDB. Además, también se pretende llevar a cabo las mismas pruebas en un entorno distinto, teniendo un sistema operativo Linux como anfitrión y un clúster como máquina de pruebas. Esto con el objetivo de comparar los tiempos de respuesta en distintos sistemas operativos y ver las diferencias que existen al implementar sobre un sistema operativo diferente, así como poder observar los resultados de rendimiento al realizar la implementación sobre un clúster.

Anexos

Anexo A. Códigos para la generación de las BD y la ejecución de las pruebas

En este apartado, se presenta un enlace donde se encuentran alojados los archivos con los códigos y comandos utilizados para la generación de la BD TPC-H y TPC-E, así como los archivos que contienen los datos de cada BD.

https://drive.google.com/drive/folders/1_lC3IdVIe5mniUAalD7POHGEZZrBisk3?usp=sharing

Primeramente, se encuentran dos directorios: TPC-H y TPC-E. Cada directorio contiene un directorio más por cada SGBD (MySQL, PostgreSQL, MongoDB); a su vez, dentro de cada directorio del gestor se ubican los archivos necesarios para la creación de las BD (dss.ddl y dss.ri), así como el *script* que carga las tuplas a las tablas y un archivo con algunos otros comandos utilizados. Además, también se encuentran las consultas utilizadas por cada gestor y los *scripts* con cada tipo de particionamiento desarrollado.

Además, también se incluyen los códigos desarrollados para la ejecución de las pruebas, estos se desarrollaron en dos distintos paquetes, por lo que, dentro del directorio correspondiente, TPC-H y TPC-E, existe un paquete Java nombrado TPCH_Query y TPCE_Query, respectivamente.

Anexo B. Publicación

Derivado de este trabajo de tesis, se elaboró el artículo titulado "Caracterización de los tipos de Fragmentación en Sistemas Gestores de Bases de Datos Relacionales", el cual fue aceptado y presentado en la 9ª Jornada de Ciencia y Tecnología Aplicada 2022 (JCyTA) llevada a cabo del 16 al 18 de Noviembre del año 2022. JCyTA es un foro abierto para que estudiantes, investigadores y profesionales presenten innovaciones y resultados en las diversas ciencias de la ingeniería, celebrado por el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET).

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

Boussahoua, M., Bentayeb, F., Boussaid, O., & Kabachi, N. (2018). A Data Partitioning Optimization Approach for Distributed Data Warehouses on Column family NoSQL Systems.

Cassandra. (2021). Cassandra Documentation. Recuperado en Octubre de 2021 de https://cassandra.apache.org/doc/latest/cassandra/architecture/overview.html

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

Date, C. J. (2001). Introducción a los Sistemas de Bases de Datos. (7° ed.). México: Pearson Educación, pp. 25-46. Recuperado el 10 de Septiembre de 2021 de https://unefazuliasistemas.files.wordpress.com/2011/04/introducion-a-los-sistemas-de-bases-de-datos-cj-date.pdf

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

Elastic. (2021). Elasticsearch Guide. Recuperado en Octubre de 2021 de https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

Fraczek, K., & Plechawska-Wojcik, M. (2017). Comparative Analysis of Relational and Non-relational Databases in the Context of Performance in Web Applications. In: Kozielski S., Mrozek D., Kasprowski P., Małysiak-Mrozek B., Kostrzewa D. (eds) Beyond Databases, Architectures and Structures. Towards Efficient

Solutions for Data Analysis and Knowledge Representation. BDAS 2017. Communications in Computer and Information Science, vol. 716. Springer. DOI: https://doi.org/10.1007/978-3-319-58274-0_13

IBM. (2021). Db2 11.5. Documentation. Recuperado en Octubre de 2021 de https://www.ibm.com/docs/en/db2/11.5

Khashan, E. A., El Desouky, A. I., & Elghamrawy, S. M. (2020). A Framework for Executing Complex Querying for Relational and NoSQL Databases (CQNS). European Journal of Electrical Engineering and Computer Science. 4, 5 (Sep. 2020). DOI: https://doi.org/10.24018/ejece.2020.4.5.195

Kozma, F., & Morschheuser, T. (2019). Cloud Service Environment PostgreSQL vs. Cassandra. Software Engineering. Blekinge Institute of Technology. SE-371 79 Karlskrona Sweden. urn:nbn:se:bth-18062

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

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

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

MongoDB[1]. (2021). MongoDB Manual. Recuperado en Octubre de 2021 de https://www.mongodb.com/docs/v5.0/tutorial/getting-started/

MongoDB[2]. (2022). The MongoDB Database Tools Documentation. Recuperado en Octubre de 202 de. https://www.mongodb.com/docs/database-tools/

MongoDB[3]. (2022). *Basic Cluster Administration*. MongoDB University. Recuperado en Octubre de 2021 de https://university.mongodb.com/?tck=docs_landing

MySQL[1]. (2021). *Partitioning Types*. MySQL 8.0 Reference Manual. Recuperado en Octubre de 2021 de https://dev.mysql.com/doc/refman/8.0/en/partitioning-types.html

MySQL[2]. (2021). MySQL 8.0 Reference Manual. Recuperado en Octubre de 2021 de https://dev.mysql.com/doc/refman/8.0/en/

Nam, Y.-M., Han, D., & Kim, M.-S. (2019). A parallel query processing system based on graph-based database partitioning. Information Sciences, 480, 237–260. DOI: https://doi.org/10.1016/j.ins.2018.12.031

Oracle México. (2021). ¿Qué es una base de datos? Recuperado el 10 de Septiembre de 2021 de https://www.oracle.com/mx/database/what-is-database

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

PostgreSQL[1]. (2021). *Table Partitioning.* Chapter 5. Data Definition. Recuperado en Octubre de 2021 de https://www.postgresql.org/docs/14/ddl-partitioning.html

PostgreSQL[2]. (2021). *PostgreSQL 14.1 Documentation*. The PostgreSQL Global Development Group. Recuperado en Octubre de 2021 de https://www.postgresql.org/docs/14/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

Ramez, E., & Shamkant, B. N. (2007). Fundamentos de Sistemas de Bases de Datos. (5° ed.). Madrid: Pearson Educación, p. 4. Recuperado el 10 de Septiembre de 2021, de https://es.pdfdrive.com/elmasri-ramez-navathe-shamkant-b-2007-fundamentos-de-sistemas-de-bases-de-datos-5a-ed-madrid-pearson-educac-e186403211.html

Redis. (2021). Documentation. Recuperado en Octubre de 2021 de https://redis.io/documentation

Romero, O., Herrero, V., Abelló, A., & Ferrarons, J. (2015). Tuning small analytics on Big Data: Data partitioning and secondary indexes in the Hadoop ecosystem. Information Systems, 54, 336–356. DOI: https://doi.org/10.1016/j.is.2014.09.005

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 BENCHMARK™ E*. Version 1.14.0. Recuperado en Diciembre de 2021 de https://www.tpc.org/tpce/

Serlin, O., Sawyer, T., & Gray, J. (2021). *TPC BENCHMARK™ H*. Revision 3.0.0. Recuperado en Diciembre de 2021 de https://www.tpc.org/tpch/

SQLite. (2021). Documentation. Recuperado en Octubre de 2021 de https://www.sqlite.org/docs.html

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

Sun, L., Franklin, M. J., Wang, J., & Wu, E. (2016). Skipping-oriented partitioning for columnar layouts. Proceedings of the VLDB Endowment, 10(4), 421–432. DOI: https://doi.org/10.14778/3025111.3025123

Tamer Özsu, M., & Valduriez, P. (2020). Principles of Distributed Database Systems (Springer (Ed.); Fourth). https://doi.org/10.1007/978-3-030-26253-2_13