

Particionamiento de Tablas en MySQL

Freddy Abad ,Esteban Vizhñay
Facultad de Ingeniería, Universidad de Cuenca
Cuenca, Ecuador
{freddy.abadl,esteban.vizhñay}@ucuenca.edu.ec

Resumen. - En el mundo empresarial, la optimización de recursos es una etapa muy necesaria para la escalabilidad de sus servicios. El particionamiento de bases de datos densamente pobladas, otorga múltiples ventajas y beneficios. Una empresa que busque escalar sus servicios e implementen las diversas estrategias de particionamiento obtienen plataformas tecnológicas robustas para la alta concurrencia, disminución de tiempos en el mantenimiento de sus equipos y plataformas, entre otros beneficios. En un caso práctico, este informe busca explicar teóricamente las estrategias que se pueden implementar en diversos tipos de particionamientos de las bases de datos. Adicionalmente, explica la implementación de estas bases de datos utilizando el SGBD (Sistema Gestor de Base de Datos) MySQL. Los resultados permiten abstraer al lector en que situaciones practicas utilizar cada tipo de particionamiento.

Palabras clave: SGBD, Particionamiento MySQL, Bases de Datos Distribuidas, DML, DDL.

I. INTRODUCCIÓN

Al construir un sistema de Inteligencia de negocios con su respectivo Data Warehouse, uno de los objetivos es la velocidad a la hora de obtener información. Si existe la posibilidad de tener la tabla “troceada” en segmentos más pequeños seguramente aumentará el rendimiento y la velocidad del sistema. La partición de tablas se hace normalmente por razones de mantenimiento, rendimiento o gestión [1].

Para el caso de estudio Mysql, este permite distribuir tablas individuales donde cada parte es almacenada como tablas independientes en diferentes ubicaciones. Para lograr esto es necesario que se indique una regla la cual es seleccionada por el usuario que realiza la división lo cual es llamado como una función de particionamiento [2]. Esta puede ser un conjunto de rangos o lista de valores, una función hash, etc.

Existen tres tipos de partición que están presentes en MySQL [3]:

1. Partición Horizontal
 - a. Una estrategia de particionamiento que divide las tablas a nivel de filas, donde cada partición será un subconjunto de columnas de la tabla original.
2. Partición Vertical
 - a. Una estrategia de particionamiento que divide las tablas a nivel de columnas, donde se forman pequeños subconjuntos que representan columnas de interés.
3. Partición Híbrida.
 - a. Una estrategia de particionamiento que divide las tablas a nivel de filas y columnas. Donde se realiza las dos particiones antes mencionadas, el orden en que se los aplique dependerá de las necesidades de cada usuario.

El documento describe los tipos de particionamiento existentes en el Gestor de Base de Datos MySQL. Entre estos tipos, se citarán algunos ejemplos prácticos con su sintaxis estándar, lo cual mejore el entendimiento del lector.

En este apartado se explica los diferentes tipos de particionamiento que existen para MySQL.

1. Particionamiento Horizontal

El particionamiento horizontal ataca a una tabla a nivel de filas, donde el SGBD para este caso MySQL realiza el tratamiento de la misma como si fuera una tabla tradicional cada partición [4].

Las estrategias disponibles para este tipo de particionamiento son:

- **Range**

Distribuye los datos basándose en rangos, los cuales deben especificar un campo base, para poder determinar en cuál partición se debe guardar la fila. Es necesario especificar la cláusula **VALUES LESS THAN** el cual es un límite superior no inclusivo de la misma. Ejm: “**VALUES LESS THAN 5**”, significa que cualquier valor menor a 5 va a ser almacenado. Además está disponible el uso de la cláusula **MAXVALUE** para representar el valor más grande posible para el campo, por lo que cualquier valor que sobrepase la última partición definida, será almacenado en la partición que utilice esa cláusula. La sintaxis básica y un ejemplo de aplicación se puede observar en la figura 1 y figura 2 [5]. Este ejemplo práctico, permite notar como se puede dividir una tabla de ventas de manera histórica, por años.

```
CREATE TABLE table_name
PARTITIONED BY RANGE COLUMNS(column_list) (
  PARTITION partition_name VALUES LESS THAN (value_list)[,
  PARTITION partition_name VALUES LESS THAN (value_list)][,
  ...]
)
```

Figura 1: Sintaxis básica para Range

II. TIPOS DE PARTICIONAMIENTO

ventas_range				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12-12-19	4	12.5
3	5	12-07-19	5	2.1
2	3	12-06-20	1	13.2
4	6	13-09-21	7	3.5

ventas_2019				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12-12-19	4	12.5
3	5	12-07-19	5	2.1

ventas_2020				
id_venta	id_producto	timestamp	cantidad	precio
2	3	12-06-20	1	13.2

ventas_undef				
id_venta	id_producto	timestamp	cantidad	precio
4	6	13-09-21	7	3.5

Figura 2: Ejemplo práctico de la estrategia de Particionamiento Range

- **List**

Similar a la partición de rango de muchas maneras. Cada partición debe definirse explícitamente y no por un rango [6]. La sintaxis básica y un ejemplo de aplicación se puede observar en la figura 3 y figura 4. Este ejemplo práctico representa el particionamiento horizontal según los tipos de alimentos, ya sea por alimentos o por verduras, agrupados mediante una lista de valores que representan estos tipos de alimentos.

```
CREATE TABLE table_name
PARTITIONED BY LIST (column) (
  PARTITION partition_name VALUES IN (value_list)[,
  PARTITION partition_name VALUES IN (value_list)][,
  ...]
)
```

Figura 3: Sintaxis básica para List

ventas_lista				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5
3	1	12/7/2019	5	2.1
2	5	12/6/2020	1	13.2
4	6	13/9/2021	7	3.5

ventas_fruta				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5
3	1	12/7/2019	5	2.1

ventas_verdura				
id_venta	id_producto	timestamp	cantidad	precio
2	5	12/6/2020	1	13.2
4	6	13/9/2021	7	3.5

Figura 4: Ejemplo práctico de la estrategia de Particionamiento List

- **Hash**

Utilizada principalmente para garantizar una distribución uniforme de los datos entre un número predeterminado de particiones [7]. La sintaxis básica y un ejemplo de aplicación se puede observar en la figura 5 y figura 6. Este ejemplo práctico representa el particionamiento de

una tabla ventas, según un hash de su clave principal, en este caso se distribuye de manera equitativamente, sin embargo si se particiona mediante otro id, no se notara equivalente en número (n% de registros a cada n partición), sino su agrupamiento y posterior distribución.

```
CREATE TABLE table_name
PARTITION BY HASH(column)
PARTITIONS number;
```

Figura 5: Sintaxis básica para Hash

ventas_hash				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5
3	1	12/7/2019	5	2.1
2	5	12/6/2020	1	13.2
4	6	13/9/2021	7	3.5

ventas_hash_0				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5
4	6	13/9/2021	7	3.5

ventas_hash_1				
id_venta	id_producto	timestamp	cantidad	precio
3	1	12/7/2019	5	2.1
2	5	12/6/2020	1	13.2

Figura 6: Ejemplo práctico de la estrategia de Particionamiento Hash

- **Key**

Similar a la estrategia HASH, pero el SGBD es quien suministra la función que determina la partición a utilizar, la tabla debe contar con una llave primaria o un índice único. La sintaxis básica y un ejemplo de aplicación se puede observar en la figura 7 y figura 8. Este ejemplo práctico representa una tabla ventas que se particiona según su id principal, para este tipo de particionamiento es necesario que cada tabla tenga una clave principal, para que el SGBD pueda particionar.

```
CREATE TABLE table_name
PARTITION BY KEY(column_list)
PARTITIONS number;
```

Figura 7: Sintaxis básica para Key

ventas_key				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5
3	1	12/7/2019	5	2.1
2	5	12/6/2020	1	13.2
4	6	13/9/2021	7	3.5

ventas_key_0				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5

ventas_key_1				
id_venta	id_producto	timestamp	cantidad	precio
3	1	12/7/2019	5	2.1
2	5	12/6/2020	1	13.2
4	6	13/9/2021	7	3.5

Figura 8: Ejemplo práctico de la estrategia de Particionamiento Key

Si no hay ninguna clave principal pero hay una clave única, la clave única se utiliza para la clave de partición.[8]

- **Compuesto**

Estrategia de doble particionado, aplicando cualquier combinación de las estrategias anteriores [9]. Un ejemplo de aplicación se puede observar en la figura 9. Este particionamiento se realiza en dos fases, la tabla ventas, se divide por estrategia range, agrupando los datos históricos por año, a estas particiones subdivide equitativamente por años, es decir se tendrán dos particiones equitativas por año de un universo de registros de la tabla principal.

ventas_compuesto				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5
3	1	12/12/2019	5	2.1
2	5	12/6/2020	1	13.2
4	6	13/9/2021	7	3.5

ventas_compuesto_0				
id_venta	id_producto	timestamp	cantidad	precio
1	2	12/12/2019	4	12.5
3	1	12/12/2019	5	2.1

ventas_compuesto_1				
id_venta	id_producto	timestamp	cantidad	precio
2	5	12/6/2020	1	13.2

ventas_compuesto_2				
id_venta	id_producto	timestamp	cantidad	precio
4	6	13/9/2021	7	3.5


Figura 9: Ejemplo práctico de la estrategia de Particionamiento Compuesto

2. Particionamiento Vertical

Esta estrategia de particionamiento de tablas, se realiza con el fin de optimizar las consultas DML (Select, Update, Delete, Insert). Este tipo de particionamiento considera que una tabla normalmente tiene ciertas columnas que se consultan, insertan, modifican o eliminan con mayor frecuencia. Esta prevención de tratamiento a la table permite optimizar las consultas a la base de datos, disminuyendo considerablemente la extensión de queries.

Un ejemplo práctico para entender, es el planteado en la figura 10, la tabla usuario, cumple con los estándares de normalización, sin embargo, al momento de realizar modificación es más probable que un usuario actualice su contraseña o su correo electrónico con el cual ingresa a un sistema informático. Teniendo en mente este caso, se puede dividir la tabla verticalmente, y distribuir la información por ejemplo para un caso de modificación.

usuario			
id_usuario	nombre	correo_electronico	contrasena
01DE	Juan Perez	juan_perez@outlook.com	yu6745yert34532
02ES	Jose Zambrano	jose_zambrano@outlook.com	xasfd6543
03SW	Rosa Quizhpe	rosa_quizhpe@outlook.com	asfasd951



usuario_p0		
id_usuario	nombre	correo_electronico
01DE	Juan Perez	juan_perez@outlook.com
02ES	Jose Zambrano	jose_zambrano@outlook.com
03SW	Rosa Quizhpe	rosa_quizhpe@outlook.com

usuario_p1	
id_usuario	contrasena
01DE	yu6745yert34532
02ES	xasfd6543
03SW	asfasd951

Figura 10: Ejemplo Práctico de Particionamiento Vertical.

3. Particionamiento Híbrido

El particionamiento híbrido segmenta a las tablas a nivel de “super celdas”. Esto refiere a un particionamiento horizontal (por filas) y a un particionamiento vertical (por columnas).

El particionamiento híbrido (Figura 11c) se puede realizar sin orden establecido, es decir dividir primero horizontalmente (Figura 11a) y posteriormente de manera vertical (Figura 11b), o viceversa. Esta elección, debe realizar el administrador de la base de datos, o en su defecto el arquitecto de software, tomando en cuenta las necesidades del dominio del problema al cual atacar.

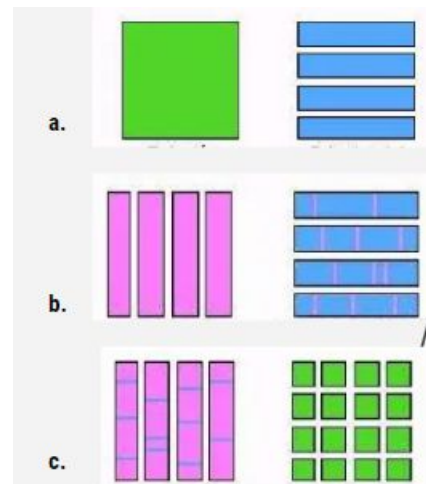


Figura 11: Particionamiento a) Horizontal, b) Vertical, c) Híbrido.

Un caso práctico para entender esta estrategia de particionamiento, se puede ver en la figura 12, en este caso, se unen los principios del particionamiento vertical y horizontal. Estos principios proponen dividir los datos históricos, y dividir este particionamiento posteriormente para las consultas DML que pueden generarse. Así una tabla usuarios, se puede dividir horizontalmente con la estrategia hash, la cual distribuye equitativamente los registros. Estas tablas particionadas se dividen nuevamente según los registros con mayor probabilidad de ser modificados.

usuario			
id_usuario	nombre	correo_electronico	contrasena
01DE	Juan Perez	juan_perez@outlook.com	yu6745yert34532
02ES	Jose Zambrano	jose_zambrano@outlook.com	xasdf6543
03SW	Rosa Quizhpe	rosa_quizhpe@outlook.com	asfasd951
04DE	Juana Dominguez	juana_dominguez@outlook.com	dfasdf89

↓

usuario_p0		
id_usuario	nombre	correo_electronico
01DE	Juan Perez	juan_perez@outlook.com
04DE	Juana Dominguez	juana_dominguez@outlook.com

usuario_p0	
id_usuario	contrasena
01DE	yu6745yert34532
04DE	dfasdf89

usuario_p1		
id_usuario	nombre	correo_electronico
03SW	Rosa Quizhpe	rosa_quizhpe@outlook.com
02ES	Jose Zambrano	jose_zambrano@outlook.com

usuario_p1	
id_usuario	contrasena
03SW	asfasd951
02ES	xasdf6543

Figura 12: Ejemplo Práctico del Particionamiento Híbrido

III. EXPERIMENTO

Como se ha mencionando anteriormente, el particionar tablas permite mejorar la velocidad de las consultas. Para este trabajo se realizó el particionamiento horizontal y vertical, ambos realizados con MySQL v 5.7.15.0.

1. Particionamiento horizontal

La implementación de cada estrategia de particionamiento se realizó en base a los ejemplos prácticos planteados en la sección II, subsecciones:

- 1 - Particionamiento Range (Figura 2)
- 2 - Particionamiento List (Figura 4)
- 3 - Particionamiento Hash (Figura 6)
- 4 - Particionamiento Key (Figura 8)
- 5 - Particionamiento Compuesto (Figura 9)

El particionamiento de la estrategia range, de un universo de 100 registros produjo particiones con los siguientes resultados de la figura 13.

Nombre Particion	TABLE_ROWS
ventas_2019	43
ventas_2020	47
ventas_2021	10

Figura 13: Resultados del Particionamiento de la Estrategia Range

El particionamiento de la estrategia list, de un universo de 100 registros produjo particiones con los siguientes resultados de la figura 14.

Nombre Particion	TABLE_ROWS
venta_fruta	41
venta_verdura	59

Figura 14: Resultados del Particionamiento de la Estrategia List

El particionamiento de la estrategia hash, de un universo de 100 registros produjo particiones con los siguientes resultados de la figura 15. En este tipo de particionamiento, no se puede nombrar las particiones, como en los casos de particionamiento tipo range y list.

Nombre Particion	TABLE_ROWS
p0	25
p1	25
p2	25
p3	25

Figura 15: Resultados del Particionamiento de la Estrategia Hash

El particionamiento de la estrategia key, de un universo de 100 registros produjo particiones con los siguientes resultados de la figura 16. En este tipo de particionamiento, al igual que la estrategia Hash, no se puede nombrar las particiones, como en los casos de particionamiento tipo range y list.

Nombre Particion	TABLE_ROWS
p0	50
p1	50

Figura 16: Resultados del Particionamiento de la Estrategia Key

El particionamiento de la estrategia compuesta, de un universo de 100 registros produjo particiones con los siguientes resultados de la figura 17.

Nombre Particion	TABLE_ROWS
ventas_c_2019	27
ventas_c_2019	16
ventas_c_2020	29
ventas_c_2020	18
ventas_c_undef	4
ventas_c_undef	6

Figura 17: Resultados del Particionamiento de la Estrategia compuesta

2. Particionamiento vertical

Como primer paso se necesita crear o utilizar una base de datos en un servidor, para este caso práctico se hará uso del programa Wampserver que por defecto nos proporciona una base de datos MySQL en un servidor local que puede ser accedido mediante PhpAdmin ver figura 18.



Figura 18: Ejemplo para Compuesto

A continuación se creará una tabla y sus columnas:

- La tabla con el nombre de Usuario y en la figura 19 se puede observar su sentencia SQL.

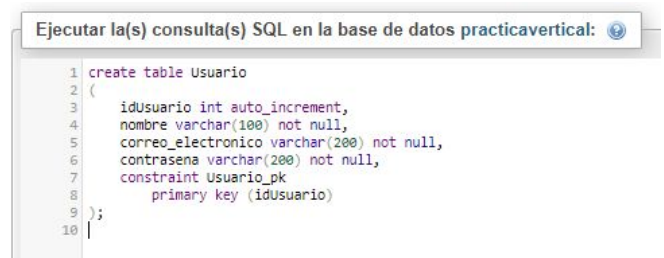


Figura 19: Sentencia para crear la tabla Usuario

Como siguiente paso es necesario crear un usuario y otorgar los permisos que se requiera, este usuario será especificado cuando se cree la base de datos local. La sentencia ejecutada para esta acción es la siguiente:

GRANT ALL ON practicavertical.* TO "estbmPC"@"%" identified by "123PC"

Hay que tener en cuenta que este usuario debe poseer únicamente los permisos necesarios ya que si se le otorga permisos que no correspondientes, la base de datos local podría alterar el contenido de la base del servidor.

Ahora en la base de datos local es requerido configurar el motor de almacenamiento FEDERATED ya que por default MySQL lo tiene deshabilitado. Para aplicar este cambio es necesario agregar debajo de la línea [mysql] la palabra **federated** del archivo my.init. de MySQL. Observar la figura 20 y la figura 21 donde la última nos muestra que Federated ha sido activado. Después de aplicar esta configuración es necesario reiniciar el servicio de MySQL ver figura 22.

```
# SERVER SECTION
# -----
#
# The following options will be read by the MySQL Server. Make sure that
# you have installed the server correctly (see above) so it reads this
# file.
#
# server_type=3
[mysql]
federated
# The next three options are mutually exclusive to SERVER_PORT below.
# skip-networking
#
# enable-named-pipe
```

Figura 20: Agregando federated a my.init de MySQL

```
mysql> show engines
->
```

Engine	Support
InnoDB	DEFAULT
MRG_MYISAM	YES
MEMORY	YES
BLACKHOLE	YES
MyISAM	YES
CSV	YES
ARCHIVE	YES
PERFORMANCE_SCHEMA	YES
FEDERATED	YES

Figura 21: Visualizando que federated está activado

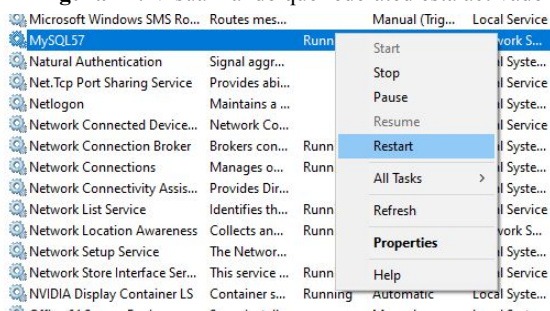


Figura 22: Reiniciando el servicio de MySQL

Como siguiente paso está en la creación de las tablas indicando que se utilizará el motor de almacenamiento FEDERATED necesario para poder comunicarnos y obtener los datos de la base del

servidor, para nuestro caso observar la figura 23 donde se muestra la sintaxis utilizada:

```
-- para crear la base local

create table UsuarioLocal (
  idUsuario int not null ,
  nombre varchar(100) not null,
  correo_electronico varchar(200) not null
) engine=FEDERATED DEFAULT CHARSET=latin1
connection='mysql://estbmPC:123PC@127.0.0.1:3307/practicavertical/usuario'
```

```
create table Usuario_P1 (
  idUsuario int not null ,
  contraseña varchar(200) not null
) engine=FEDERATED DEFAULT CHARSET=latin1
connection='mysql://estbmPC:123PC@127.0.0.1:3307/practicavertical/usuario'
```

Figura 23: Creando Tabla local

El atributo connection de la sentencia SQL requiere lo siguiente: **scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name.**

Es muy importante que las columnas tengan el mismo nombre que la tabla del servidor de lo contrario se presentará problemas. Ya con este último paso se tendrá el particionamiento vertical completo, para comprobar el funcionamiento se realizará la inserción de tuplas en la tabla usuario y en las tablas locales se ejecutará la sentencia para mostrar sus datos. Estas acciones se realizan en la figura 24 y la figura 25.

```
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Odette Vargas','Vivamus.molestie.dapibus@sed.ca','16704887')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Kendall Ward','mauris@donecfeclis.org','15700940')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Medge Jensen','semper.auctor.Mauris@egetipsum.ca','20704188')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Palmer Navarro','turpis@sapien.edu','33876710')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Burke Frederick','leo.Cras@pretiumaliquetmetus.co.uk','13612100')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Sasha Stevens','aliquet.vel@nonmagna.edu','22646150')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Abdul Heath','massa@gravidasitamet.net','37875595')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Brenda Peterson','velit.just@malesuadamalesuada.ca','47186023')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Finn Gilliam','risus@orciquis.net','10160772')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Benjamin Mitchell','neque.Morbi@Aliquam.co.uk','50400233')
INSERT INTO usuario (nombre,correo_electronico,contraseña) VALUES ('Anika Navarro','bibendum fermentum.metus@eleifendvitaerat.net','23865555')
```

Figura 24: Insertando tuplas en la base del servidor

idUsuario	nombre	correo_electronico	contraseña
1	Odette Vargas	Vivamus.molestie.dapibus@sed.ca	16704887
2	Kendall Ward	mauris@donecfeclis.org	15700940
3	Medge Jensen	semper.auctor.Mauris@egetipsum.ca	20704188
4	Palmer Navarro	turpis@sapien.edu	33876710
5	Burke Frederick	leo.Cras@pretiumaliquetmetus.co.uk	13612100
6	Sasha Stevens	aliquet.vel@nonmagna.edu	22646150
7	Abdul Heath	massa@gravidasitamet.net	37875595
8	Brenda Peterson	velit.just@malesuadamalesuada.ca	47186023
9	Finn Gilliam	risus@orciquis.net	10160772
10	Benjamin Mitchell	neque.Morbi@Aliquam.co.uk	50400233
11	Anika Navarro	bibendum fermentum.metus@eleifendvitaerat.net	23865555

idUsuario	nombre	correo_electronico
1	Odette Vargas	Vivamus.molestie.dapibus@sed.ca
2	Kendall Ward	mauris@donecfeclis.org
3	Medge Jensen	semper.auctor.Mauris@egetipsum.ca
4	Palmer Navarro	turpis@sapien.edu
5	Burke Frederick	leo.Cras@pretiumaliquetmetus.co.uk
6	Sasha Stevens	aliquet.vel@nonmagna.edu
7	Abdul Heath	massa@gravidasitamet.net
8	Brenda Peterson	velit.just@malesuadamalesuada.ca
9	Finn Gilliam	risus@orciquis.net
10	Benjamin Mitchell	neque.Morbi@Aliquam.co.uk
11	Anika Navarro	bibendum fermentum.metus@eleifendvitaerat.net
12	Martena Dalton	ultrices@vestibulumtempor.net
13	Christopher Shelton	enim.Suspendisse.aliquet@non.net
14	Keith Preston	urna.Vivamus@donecfeclis.org
15	Malcolm Knight	Donec.egestas.Aliquam@Integer.net
16	Candice Skinner	ac@augueid.org
17	Louis Moran	enim.Itiam@purus.net
18	Benjamin Higgins	ipsum@lobortismauris@Suspendisse.net
19	Zeph Reid	gravid@eturpis@Nulla.com
20	Philip Gay	eu@Sedneque.org
21	Rinah Burcardo	eu.euism@donecfeclis.org
22	Libby Burcardo	neque@Landitvivera.net
23	Desecrat.Hannington	luctus.vulnuptat@idcanian.co.uk

Figura 25: Visualizando en las tablas locales la información

IV. CONCLUSIONES

En el experimento realizado se debe tener en cuenta varios puntos, entre ellos, la elección de una fragmentación vertical, horizontal o híbrida dependerá del dominio de la base de datos a optimizar. Sin embargo, en datos históricos (no variantes) es recomendable la fragmentación horizontal es recomendada, mientras que para datos que se actualizan con más concurrencia se recomienda la fragmentación vertical. En casos donde se busque optimizar de mayor manera, y se cuente con los recursos en cuanto a servidores, se puede realizar la fragmentación híbrida. Estas decisiones se deben tomar en base a la preparación del personal técnico existente.

En la práctica se pudo notar que el particionamiento no es óptimo para bases de datos poco pobladas, según los editores de contenido de MySQL [5], el particionamiento es recomendable para tablas densamente pobladas (>2GB).

Una restricción, en el caso del SGBD MySQL, el particionamiento se debe realizar en versiones mayores a la 5.6.

En bases de datos, altamente pobladas, existe menor latencia en sentencias DDL-DML, posicionando al particionamiento (horizontal, vertical o híbrido) como una buena práctica de desarrollo, para que las transacciones se realicen en tiempos óptimos. A esta ventaja, se incluye la optimización en cuanto a accesibilidad de los datos, debido a la disminución del número de conexiones remotas (Por ejemplo: Las consultas realizadas a datos históricos, se realizan a una única partición). Esto conlleva a un mejor manejo del mantenimiento de las Bases de Datos, una escalabilidad ante Bases de Datos Distribuidas.

Adicionalmente, el particionamiento, en cualquiera de sus tipos, mejora el control de la trazabilidad ante auditorias, necesarias en cualquier empresa.

Finalizando, el particionamiento provee múltiples ventajas que predisponen ante una mejora de arquitecturas de software, sin embargo, para que se gocen de todas las ventajas, los particionamientos deben implementarse después de varias pruebas de rendimiento como control de calidad.

V. BIBLIOGRAFÍA

[1] "Tabla Hechos Venta. Particionado en MySql. | Dataprix TI", Dataprix.com, 2020. [Online]. Available: <https://www.dataprix.com/es/blog-it/respinosamilla/tabla-hechos-venta-particionado-mysql>. [Accessed: 04- Apr- 2020].

[2] "MySQL :: MySQL 5.7 Reference Manual :: 22.1 Overview of Partitioning in MySQL", Dev.mysql.com, 2020. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/partitioning-overview.html>. [Accessed: 04- Apr- 2020].

[3] 2020. [Online]. Available: https://www.academia.edu/25333226/Bases_de_Datos_Distribuidas. [Accessed: 04- Apr- 2020].

[4] "Investigación - Tablas Particionadas - David Villalobos C.pdf", Google Docs, 2020. [Online]. Available: <https://drive.google.com/file/d/0B5qeW3qtOVL6R2xxSWHsAHVhVU0/view>. [Accessed: 04- Apr- 2020].

[5] "MySQL :: MySQL 5.7 Reference Manual :: 22.2.1 RANGE Partitioning", Dev.mysql.com, 2020. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/partitioning-range.html>. [Accessed: 04- Apr- 2020].

[6] "MySQL :: MySQL 5.7 Reference Manual :: 22.2.2 LIST Partitioning", Dev.mysql.com, 2020. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/partitioning-list.html>. [Accessed: 04- Apr- 2020].

[7] "MySQL :: MySQL 5.7 Reference Manual :: 22.2.4 HASH Partitioning", Dev.mysql.com, 2020. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/partitioning-hash.html>. [Accessed: 04- Apr- 2020].

[8] "MySQL :: MySQL 5.7 Reference Manual :: 22.2.5 KEY Partitioning", Dev.mysql.com, 2020. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/partitioning-key.html>. [Accessed: 04- Apr- 2020].

[9] "MySQL :: MySQL 5.7 Reference Manual :: 22.2.6 Sub Partitioning", Dev.mysql.com, 2020. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/partitioning-subpartitions.html>. [Accessed: 04- Apr- 2020].