



Particionamiento de Tablas en MySQL

ALMACENAMIENTO Y MINERÍA DE DATOS

Abad L. Freddy L.

Vizhnay E. Esteban D.



Introducción

01

Tipos de Particionamiento

02

Ejemplo Práctico

03

Conclusiones

04

Referencias

05



01

Introducción



Introducción

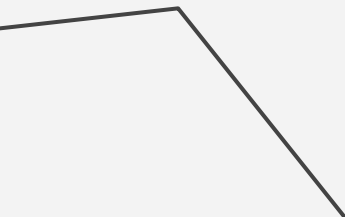


- Un objetivo principal dentro de un sistema de inteligencia de negocios implementando un datawarehouse, es la *velocidad a la hora de obtener información* de bases de datos “grandes”.
- La velocidad se puede lograr por diversas técnicas, una de ellas es el **particionado**.
- El particionado distribuye porciones de una tabla individual en diferentes segmentos conforme a unas reglas establecidas por el usuario.
- La partición de tablas se hace normalmente por razones de ***mantenimiento, rendimiento o gestión***.



02

Tipos de Particionamiento



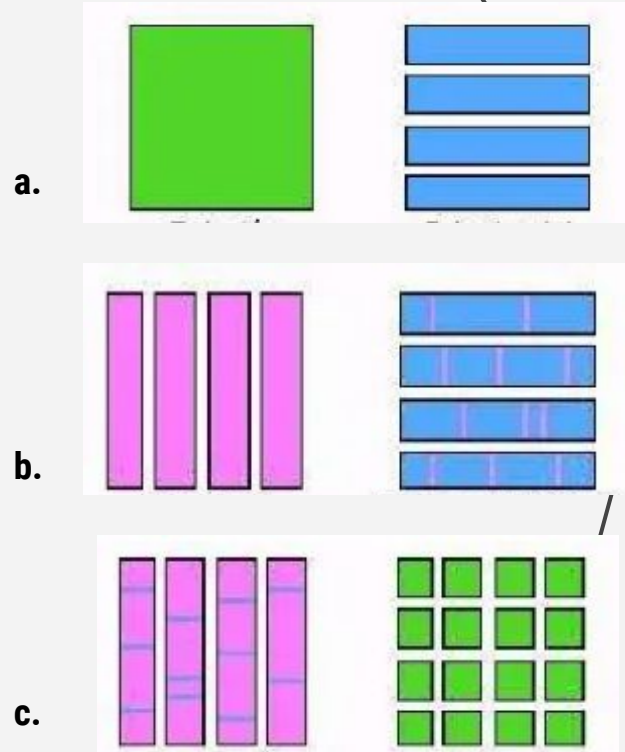
Tipos

Particionamiento

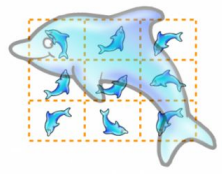
Particionamiento Horizontal: Se realiza sobre las tuplas de la relación, es decir que cada fragmento será un subconjunto de las tuplas de la relación. Divide por Filas. (Figura a).

Particionamiento Vertical: Divide la relación en un conjunto de relaciones más pequeñas tal que algunas de las aplicaciones de usuario sólo hagan uso de un fragmento. Divide por Columnas. (Figura b)

Particionamiento Híbrida: Particionamiento Horizontal sobre un Particionamiento Vertical o viceversa. Divide por columnas y filas. (Figura c)

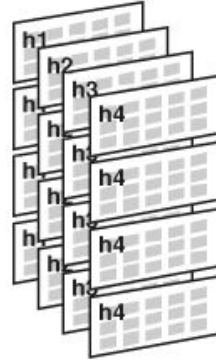


Partición Horizontal



Def: Tabla que está partida en **n partes**. El SGBD hace el tratamiento de la misma como si fuera una tabla tradicional.

Composite Partitioning
Range-Hash



Composite Partitioning
Range - List

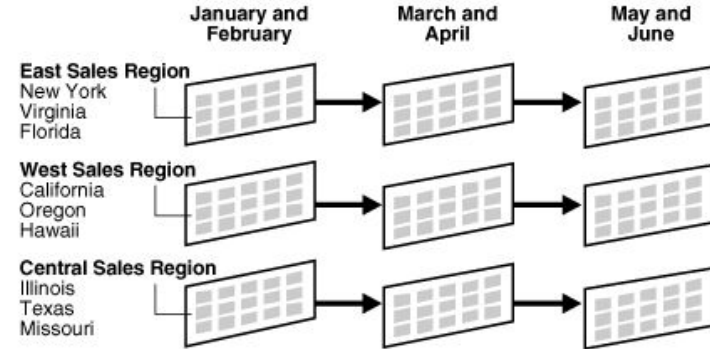
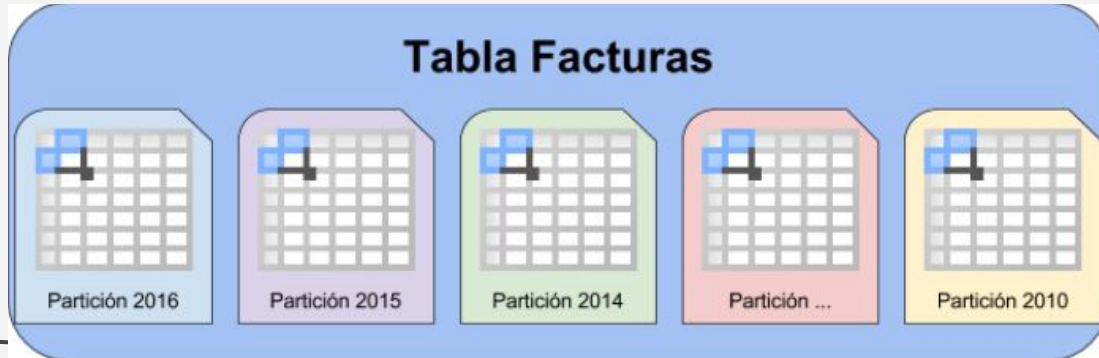
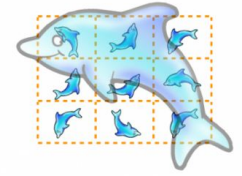


Tabla Facturas



Ejemplo: Tabla factura puede ser particionada por año.

Estrategias de particionado horizontal



Para MySQL existen las siguientes estrategias:

- RANGE
- LIST
- HASH
- KEY
- COMPUESTO

Estrategia RANGE o RANGO

Distribuye los datos basándose en rangos, los cuales deben especificar un campo base, para poder determinar en cuál partición se debe guardar la fila.

Sintaxis básica:

```
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)][,
    ...]
)
```

VALUES LESS THAN: Límite superior no inclusivo de la misma. *Ejm:* "VALUES LESS THAN 5", significa que cualquier valor menor a 5 va a ser almacenado.

MAXVALUE para representar el valor más grande posible para el campo.

Ejemplo de Estrategia RANGE

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

```
CREATE TABLE ventas_range(  
  id_venta INT,  
  id_producto INT,  
  fecha DATE,  
  cantidad REAL,  
  precio REAL  
)  
PARTITION BY RANGE (YEAR(fecha)) (  
  PARTITION `ventas_2019` VALUES LESS THAN (2020),  
  PARTITION `ventas_2020` VALUES LESS THAN (2021),  
  PARTITION `ventas_2021` VALUES LESS THAN (2022)  
);
```

Nombre Particion	TABLE_ROWS
ventas_2019	43
ventas_2020	47
ventas_2021	10

Estrategia LIST

Similar a RANGE, pero los valores están dados literalmente y no por un “rango”.

Sintaxis básica :

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

Ejemplo de Estrategia 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

```
CREATE TABLE ventas_list (  
  id_venta INT,  
  id_producto INT,  
  fecha DATE,  
  cantidad REAL,  
  precio REAL  
)  
PARTITION BY LIST(id_producto)  
(  
  PARTITION venta_fruta VALUES IN (1, 2,3,4),  
  PARTITION venta_verdura VALUES IN (5,6,7,8,9,10)  
);
```

Nombre Particion	TABLE_ROWS
venta_fruta	41
venta_verdura	59

-
-
-
-
-
-

Estrategia HASH

Utilizada principalmente para garantizar una **distribución uniforme** de los datos entre un número predeterminado de particiones.

Sintaxis básica:

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

Ejemplo de Estrategia 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

```
CREATE TABLE ventas_hash(  
    id_venta INT,  
    id_producto INT,  
    fecha DATE,  
    cantidad REAL,  
    precio REAL  
)  
PARTITION BY HASH(id_venta)  
PARTITIONS 4;
```

	Nombre Particion	TABLE_ROWS
▶	p0	25
	p1	25
	p2	25
	p3	25

-
-
-
-
-
-

Estrategia 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.

Sintaxis básica:

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

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

**** column_list es opcional**

Ejemplo de Estrategia 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

```
CREATE TABLE `ventas_key`(  
  id_venta INT PRIMARY KEY,  
  id_producto INT,  
  fecha DATE,  
  cantidad REAL,  
  precio REAL  
)  
PARTITION BY KEY()  
PARTITIONS 2;
```

	Nombre Particion	TABLE_ROWS
►	p0	50
	p1	50

Estrategia COMPUESTO (Subpartitioning)

Estrategia de doble particionado, aplicando cualquier combinación de las estrategias anteriores

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

```
CREATE TABLE ventas_compuesta(  
    id_venta INT,  
    id_producto INT,  
    fecha DATE,  
    cantidad REAL,  
    precio REAL  
)  
  
PARTITION BY RANGE (YEAR(fecha))  
SUBPARTITION BY HASH(TO_DAYS(fecha))  
(PARTITION `ventas_c_2019` VALUES LESS THAN (2020) (  
    SUBPARTITION s0,  
    SUBPARTITION s1  
),  
  
    PARTITION `ventas_c_2020` VALUES LESS THAN (2021)(  
    SUBPARTITION s2,  
    SUBPARTITION s3  
),  
  
    PARTITION `ventas_c_undef` VALUES LESS THAN MAXVALUE(  
    SUBPARTITION s4,  
    SUBPARTITION s5  
)  
);
```

	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

Particionamiento Vertical

Estrategia de particionamiento de tablas, con el fin de optimizar consultas. Una tabla normalmente tiene ciertas columnas que se consultan - modifican con mayor frecuencia. La Partición Vertical está orientado a optimizar estas consultas, disminuyendo la extensión de queries.

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



usuario_p0			usuario_p1	
id_usuario	nombre	correo_electronico	id_usuario	contrasena
01DE	Juan Perez	juan_perez@outlook.com	01DE	yu6745yert34532
02ES	Jose Zambrano	jose_zambrano@outlook.com	02ES	xasdf6543
03SW	Rosa Quizhpe	rosa_quizhpe@outlook.com	03SW	asfasd951

Particionamiento Híbrido

Particiona las tablas a nivel de celdas (filas y columnas). Ejemplo: Tabla Usuario se particiona horizontalmente con la estrategia hash mediante `id_usuario` y posteriormente se particiona verticalmente para las columnas recurrentes a modificar.

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			usuario_p0	
id_usuario	nombre	correo_electronico	id_usuario	contrasena
01DE	Juan Perez	juan_perez@outlook.com	01DE	yu6745yert34532
04DE	Juana Dominguez	juana_dominguez@outlook.com	04DE	dfasdf89
usuario_p1			usuario_p1	
id_usuario	nombre	correo_electronico	id_usuario	contrasena
03SW	Rosa Quizhpe	rosa_quizhpe@outlook.com	03SW	asfasd951
02ES	Jose Zambrano	jose_zambrano@outlook.com	02ES	xasdf6543

03

Ejemplo práctico



04

Conclusiones





Conclusiones

- Restricción de fragmentación en MySQL (versiones mayores a la 5.6).
- La elección de una fragmentación vertical, horizontal o híbrida dependerá del dominio de la base de datos a optimizar, aunque en datos históricos (no variantes) la f. Horizontal es recomendada, mientras que para datos que se actualizan con más concurrencia se recomienda la f. Vertical.
- Particionamiento optimiza el rendimiento en tablas densamente pobladas (> 2GB).



Conclusiones

- Particionamiento optimiza accesibilidad de los datos y disminuye el número de conexiones remotas (Ejm: Consultas de año ataca a una sola partición).
- Optimiza mantenimiento de la Base de Datos.
- Aplica para Base de Datos Distribuidas.
- Particionamiento Horizontal ayuda mantener por separado los datos históricos a los datos que se insertan.
- Mejora el control de la trazabilidad. (LOGs - Auditoria).
- Se debe ejecutar pruebas de rendimiento como control de calidad.

05

Referencias

A 5x2 grid of dots, consisting of two vertical columns of five dots each.

- Tabla Hechos Venta. Particionado en MySQL. | Dataprix TI. (2020). Retrieved 29 March 2020, from <https://www.dataprix.com/es/blog-it/respinosamilla/tabla-hechos-venta-particionado-mysql>
- MySQL :: MySQL 8.0 Reference Manual :: 23 Partitioning. (2020). Retrieved 29 March 2020, from <https://dev.mysql.com/doc/refman/8.0/en/partitioning.html>
- Villalobos C, D. (2016). Universidad CENFOTEC Maestría en Tecnología de Bases de Datos. Retrieved 29 March 2020, from <https://drive.google.com/file/d/0B5qeW3qtOVL6R2xxSWhSaHVhVU0/view>
<https://www.youtube.com/watch?v=pOAKQAJp07M&t=622s>
- MySQL Partitioning - w3resource. (2020). Retrieved 29 March 2020, from <https://www.w3resource.com/mysql/mysql-partition.php>
- Particionamiento en MySQL, Valdez Juan (2020). Retrieved 29 March 2020, from https://www.academia.edu/25333226/Bases_de_Datos_Distribuidas
- Particionamiento Vertical. Vazquez Angel. Retrieved 29 March 2020, from <https://es.slideshare.net/AngelVazquez2/fragmentacion-vertical-mysql>
- Partition MySQL, Editores Documentacion Oficial. Retrieved 29 March 2020, from <https://dev.mysql.com/doc/refman/8.0/en/partitioning-overview.html>
- Particionamiento Vertical, Horizontal, Hibrido. Mateos Victor. Retrieved 29 March 2020, from <https://www.youtube.com/watch?v=VJZ3JFI3J44>

Abstract geometric lines in dark gray, forming a large, irregular shape on the left side of the slide. The lines extend from the top and bottom edges towards the center, creating a sense of depth and framing the text.

iGracias!

¿Preguntas?