MySQL Partition

https://dev.mysql.com/doc/refman/8.0/en/alter-table-partition-operations.html
http://download.nust.na/pub6/mysql/tech-resources/articles/performance-partitioning.html#:~:text=There%20are%20a%20number%20of,necessary%20partitions%20during%20query%20execution.

http://mysql.rjweb.org/doc.php/ricksrots

1. **Horizontal Partitioning** - this form of partitioning segments table rows so that distinct groups of physical row-based datasets are formed that can be addressed individually (one partition) or collectively (one-to-all partitions). All columns defined to a table are found in each set of partitions so no actual table attributes are missing. An example of horizontal partitioning might be a table that contains ten years worth of historical invoice data being partitioned into ten distinct partitions, where each partition contains a single year's worth of data.
2. **Vertical Partitioning** - this partitioning scheme is traditionally used to reduce the width of a target table by splitting a table vertically so that only certain columns are included in a particular dataset, with each partition including all rows. An example of vertical partitioning might be a table that contains a number of very wide text or BLOB columns that aren't addressed often being broken into two tables that has the most referenced columns in one table and the seldom-referenced text or BLOB data in another.

4. **Range** - this partitioning mode allows a DBA to specify various ranges for which data is assigned. For example, a DBA may create a partitioned table that is segmented by three partitions that contain data for the 1980's, 1990's, and everything beyond and including the year 2000.
5. **Hash** - this partitioning mode allows a DBA to separate data based on a computed hash key that is defined on one or more table columns, with the end goal being an equal distribution of values among partitions. For example, a DBA may create a partitioned table that has ten partitions that are based on the table's primary key.
6. **Key** - a special form of Hash where MySQL guarantees even distribution of data through a system-generated hash key.
7. **List** - this partitioning mode allows a DBA to segment data based on a pre-defined list of values that the DBA specifies. For example, a DBA may create a partitioned table that contains three partitions based on the years 2004, 2005,

and 2006.
8. **Composite** - this final partitioning mode allows a DBA to perform sub-partitioning where a table is initially partitioned by, for example range partitioning, but then each partition is segmented even further by another method (for example, hash).

Show information about partition
```sql
SHOW CREATE TABLE SIGNALS
```

Delete

All partitions
```sql
ALTER TABLE t1 REMOVE PARTITIONING;
```

One Partition
```sql
ALTER TABLE t1 DROP PARTITION p0, p1;
```

Drop a partition
The [`DISCARD PARTITION ... TABLESPACE`](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html "13.1.9 ALTER TABLE Statement") and [`IMPORT PARTITION ... TABLESPACE`](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html "13.1.9 ALTER TABLE Statement") options extend the [Transportable Tablespace](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_transportable_tablespace "transportable tablespace") feature to individual `InnoDB` table partitions. Each `InnoDB` table partition has its own tablespace file (`.ibd` file). The [Transportable Tablespace](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_transportable_tablespace "transportable tablespace") feature makes it easy to copy the tablespaces from a running MySQL server instance to another running instance, or to perform a restore on the same instance. Both options take a comma-separated list of one or more partition names. For example:

```sql
ALTER TABLE t1 DISCARD PARTITION p2, p3 TABLESPACE;
```

You can move this partition file .ibd to another server and import

Import Tablespace Partition

```sql
ALTER TABLE t1 IMPORT PARTITION p2, p3 TABLESPACE;
```

Create
ALTER TABLE `SIGNALS`

PARTITION BY RANGE(`id`)

(

 PARTITION p_2018 VALUES less than (56839003),

 PARTITION p_2019 VALUES less than (61466903),

 PARTITION p_others VALUES LESS THAN MAXVALUE

);

Reorganize Partition
```sql
ALTER TABLE SIGNALS
REORGANIZE PARTITION p_others INTO (
   PARTITION p_2020 VALUES less than (91466903),
   PARTITION p_others_2 VALUES LESS THAN MAXVALUE
);
```

01-02-2017: 1483333200

01-02-2018: 1514869200

01-02-2019: 1546405200

01-02-2020: 1577941200

find first id from timestamp

Find next year starting timestamp = previous timestamp + 31536000

1546405200 (2019) + 31536000 = 1577941200 (2020) + 31536000 = 1609477200 (2021) + 31536000

SELECT `id`,`symbol`,`date`,`timestamp`from SIGNALS where `timestamp`=1577941200 order by `id` asc;

```sql
ALTER TABLE tbl REMOVE PARTITIONING;
```

```sql
ALTER TABLE registrations
REORGANIZE PARTITION p0 INTO (
   PARTITION p0 VALUES LESS THAN (10000),
   PARTITION p0 VALUES LESS THAN (20000)
);
```

SELECT

Rows of each partitions using `information_schema`

```sql
SELECT PARTITION_ORDINAL_POSITION, TABLE_ROWS, PARTITION_METHOD
      FROM information_schema.PARTITIONS
      WHERE TABLE_SCHEMA = 'stock_bca' AND TABLE_NAME = 'SIGNALS';
```

```

### Result
 10636.040 seconds, 89702725 rows affected
```