



Partitioning in MySQL 5.1

Matthew Montgomery
Sr. Support Engineer
MySQL Support, Database Group
matt.montgomery@sun.com

Agenda

- Partitioning Overview
- Partition Types
- Partition Management
- Partition Pruning
- Restrictions & Limitations
- Sharding Overview
- Sharding Techniques
- Sharding in 3rd party frameworks

Partitioning Overview

- Partitioning allows you to **distribute** portions of individual **tables** across multiple files **according to** a set of **rules**.
- MySQL 5.1 supports **horizontal partitioning**. That is, different rows of a table may be assigned to different partitions. MySQL 5.1 does not support **vertical partitioning**, in which different columns of a table are assigned to different partitions. There are no plans to add this. This is easily worked around using VIEWS.
- The user-selected rule by which the division of data is accomplished is known as a ***partitioning function***.
- This expression can be either an **integer column** value, or a **function** acting on one or more column values and **returning an integer**.
- List of supported partitioning functions is found:
<http://dev.mysql.com/doc/refman/5.1/en/partitioning-limitations-functions.html>

NULLs

- Partitioning in MySQL does nothing to disallow NULL as the value of a partitioning expression, whether it is a column value or the value of a user-supplied expression
- MySQL treats NULL as being less than any non-NULL value, just as ORDER BY does.

Partitioning Types

- RANGE Partitioning
- LIST Partitioning
- HASH Partitioning
- KEY Partitioning
- Subpartitioning

RANGE Partitioning

- Range partitioned table is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range.
- Ranges should be **contiguous** but **not overlapping**, and are defined using the **VALUES LESS THAN** operator

RANGE Partitioning Example 1

```
CREATE TABLE students (  
    id INT NOT NULL,  
    fname VARCHAR(30),  
    lname VARCHAR(30),  
    birthday DATE,  
    grade TINYINT NOT NULL  
)  
PARTITION BY RANGE (grade) (  
    PARTITION elementary VALUES LESS THAN (5),  
    PARTITION middle VALUES LESS THAN (9),  
    PARTITION high VALUES LESS THAN MAXVALUE  
);
```

RANGE Partitioning Example 2

```
CREATE TABLE students (  
    id INT NOT NULL,  
    fname VARCHAR(30),  
    lname VARCHAR(30),  
    birthday DATE,  
    grade TINYINT NOT NULL  
)  
PARTITION BY RANGE ( YEAR(birthday) ) (  
    PARTITION p0 VALUES LESS THAN (1980),  
    PARTITION p1 VALUES LESS THAN (1990),  
    PARTITION p2 VALUES LESS THAN (2000)  
);
```

LIST Partitioning

- List partitioning in MySQL is similar to range partitioning. The chief difference is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values. This is done by using `PARTITION BY LIST(expr)`

LIST Partitioning Example

```
CREATE TABLE students (  
    id INT NOT NULL,  
    fname VARCHAR(30),  
    lname VARCHAR(30),  
    birthday DATE,  
    grade TINYINT NOT NULL,  
)  
PARTITION BY LIST ( MONTH(birthday) ) (  
PARTITION spring VALUES IN (3, 4, 5),  
PARTITION summer VALUES IN (6, 7, 8),  
PARTITION fall VALUES IN (9, 10, 11),  
PARTITION winter VALUES IN (12, 1, 2)  
);
```

HASH Partitioning 1/2

- Partitioning by HASH is used primarily to ensure an even distribution of data among a predetermined number of partitions.
- With RANGE or LIST partitioning, you must specify explicitly into which partition a row with a given column value(s) is to be stored
- With HASH you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

HASH Partitioning 2/2

- HASH Partitioning uses a **modulus** algorithm to determine the partition
- The expression used as the partitioning function should return a **varying** but **deterministic** result.
- For example, MONTH(birthday) and 4 partitions
 $\text{MOD}(\text{MONTH}('2005-09-01'), 4) = \text{MOD}(9, 4) = 1$

LINEAR HASH Partitioning

- LINEAR HASH partitioning uses a **power-of-two** algorithm.
- The advantage in partitioning by linear hash is that the adding, dropping, merging, and splitting of partitions is made **much faster**, which can be beneficial when dealing with tables containing extremely large amounts (terabytes) of data.
- The disadvantage is that data is **less** likely to be **evenly distributed** between partitions as compared with using regular hash partitioning.

HASH Partitioning Example

```
CREATE TABLE students (  
    id INT NOT NULL,  
    fname VARCHAR(30),  
    lname VARCHAR(30),  
    birthday DATE,  
    grade TINYINT NOT NULL  
)  
PARTITION BY LINEAR HASH ( DAY(birthday) )  
PARTITIONS 30;
```

KEY Partitioning

- Partitioning by key is similar to partitioning by hash, except that where hash partitioning employs a user-defined expression, the hashing function for key partitioning is supplied by the MySQL server.
- NDB Cluster uses MD5()
- For other storage engines MySQL uses its own internal hashing function which is based on PASSWORD()
- PRIMARY KEY is used by default. UNIQUE key will be used if no PRIMARY KEY exists.

KEY Partitioning Example

```
CREATE TABLE students (  
    id INT NOT NULL PRIMARY KEY,  
    fname VARCHAR(30),  
    lname VARCHAR(30),  
    birthday DATE,  
    grade TINYINT NOT NULL  
)  
PARTITION BY [LINEAR] KEY()  
PARTITIONS 10;
```

Subpartitioning

- It is possible to subpartition tables that are partitioned by RANGE or LIST. Subpartitions may use either HASH or KEY partitioning. This is also known as composite partitioning.
- Each partition must have the same number of subpartitions.
- It is also possible to define subpartitions explicitly using SUBPARTITION clauses to specify options for individual subpartitions.
- SUBPARTITIONS accept DATA DIRECTORY and INDEX DIRECTORY options.

Subpartitioning Example 1/2

```
CREATE TABLE students (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  birthday DATE,  
  grade TINYINT NOT NULL,  
  PRIMARY KEY (id,birthday)  
)  
PARTITION BY LIST ( MONTH(birthday) )  
SUBPARTITION BY KEY(id) (  
  PARTITION spring VALUES IN (3, 4, 5) (  
    SUBPARTITION p0a  
      DATA DIRECTORY = '/disk0'  
      INDEX DIRECTORY = '/disk0',  
    SUBPARTITION p0b  
      DATA DIRECTORY = '/disk1'  
      INDEX DIRECTORY = '/disk1' ),
```

(Cont...)

Subpartitioning Example 2/2

```
PARTITION summer VALUES IN (6, 7, 8) (  
  SUBPARTITION p1a  
    DATA DIRECTORY = '/disk0'  
    INDEX DIRECTORY = '/disk0',  
  SUBPARTITION p1b  
    DATA DIRECTORY = '/disk1'  
    INDEX DIRECTORY = '/disk1' ),  
PARTITION fall VALUES IN (9, 10, 11)(  
  SUBPARTITION p2a  
    DATA DIRECTORY = '/disk0'  
    INDEX DIRECTORY = '/disk0',  
  SUBPARTITION p2b  
    DATA DIRECTORY = '/disk1'  
    INDEX DIRECTORY = '/disk1' ),  
PARTITION winter VALUES IN (12, 1, 2) (  
  SUBPARTITION p3a  
    DATA DIRECTORY = '/disk0'  
    INDEX DIRECTORY = '/disk0',  
  SUBPARTITION p3b  
    DATA DIRECTORY = '/disk1'  
    INDEX DIRECTORY = '/disk1' )
```

);

Partition Management commands

- ALTER TABLE...
 - ADD PARTITION (*definition*) or PARTITIONS *number*
 - DROP PARTITION *partition_names*
 - COALESCE PARTITION *number*
 - REORGANIZE PARTITION *names* INTO (*definitions*)
 - ANALYZE/CHECK/OPTIMIZE/REBUILD/REPAIR PARTITION *partition_names*
 - REMOVE PARTITIONING

Management of RANGE and LIST Part...

- Remove specific partitions using
 - ALTER TABLE... DROP PARTITION *name*
- Add partitions using:
 - ALTER TABLE...ADD PARTITION (PARTITION p4 VALUES IN LIST (4,6,8));
 - ALTER TABLE... ADD PARTITON (PARTITION p4 VALUES LESS THAN *value*

Management of HASH and KEY Part...

- To reduce the number of KEY or HASH partitions in a table by N use:
 - ALTER TABLE... COALESCE PARTITION N ;
- This takes the data from the last N partitions and evenly distributes it over the remaining partitions.
- Increase the number of partitions using:
 - ALTER TABLE students ADD PARTITION PARTITIONS N

Maintenance of Partitions

- Defragment partitions using:
 - ALTER TABLE... REBUILD PARTITION *name*;
- ANALYZE PARTITION, CHECK PARTITION, OPTIMIZE PARTITION, and REPAIR PARTITION were also introduced in MySQL 5.1.5 but subsequently removed in 5.1.24. (Bug#20129)
- Must use **myisamchk** for partitioned MyISAM tables.

Information about Partitions.

- SHOW CREATE TABLE
- SHOW TABLE STATUS
- Query information_schema.PARTITIONS
- See partitions used by queries using:
 - EXPLAIN PARTITIONS SELECT...
 - Cannot use PARTITIONS and EXTENDED keywords together in EXPLAIN SELECT...

Restrictions & Limitations

- All partitions must use the same engine. (fixing)
- A PRIMARY KEY must include all columns in the table's partitioning function.
- ARCHIVE, BLACKHOLE, CSV, FEDERATED and MERGE tables cannot be partitioned.
- NBD supports only [LINEAR] KEY partitioning.
- When performing an upgrade, tables which are partitioned by KEY and which use any storage engine other than NDBCLUSTER must be dumped and reloaded.
- Access to partitions is sequential.