CONTENTS INCLUDE:

Database Partitioning with MySQL

Improving Performance, Availability, and Manageability

By Narayana Maruvada

MYSQL – THE MOST CHOSEN DATABASE

MySQL, the world's most popular open-source database management system, encompasses key attributes such as high performance, manageability, scalability, availability and ease of use. It has become the default database for building any new generation applications over any technology stack. Additionally, its capability to run on any platform has made it a reliable option to use because of its flexibility and control over the applications.

This reference card provides an overview of the MySQL database partitioning techniques and how these techniques lead to operational excellence.

WHAT IS DATABASE PARTITIONING?

Partitioning is a database design technique with details as follows

Major Abilities of Partitioning
1. Splits large database into smaller and more manageable ones.
2. Simplifies the overall data management and operations.
3. Enhances the overall performance of the database.
4. Assists the MySQL Optimizer.

Hot Tip MySQL Optimizer typically contains the set of routines that decide what execution path the database server should take for executing the queries.

DATABASE PARTITIONING BASICS

The two very basic database partitioning practices are as follows.

Technique	Description
Horizontal	Partitions a big table based on the number of rows.
Vertical	Partitions a table based on the different columns

DATABASE PARTITIONING IN MYSQL

MySQL implements the database partitioning feature through user-defined partitioning.

Advantages
1. Allows for more control over the records stored in the database.
2. Specifies the partition in which a record is stored
3. Default is based on a specific value

To determine whether the MySQL server supports partitioning features or not, issues a SHOW VARIABLES statement at the mysql prompt, as show below.

```
mysql> SHOW VARIABLES LIKE '%partition%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| have_partitioning | YES   |
+-------------------+-------+
1 row in set (0.00 sec)
```

Since the variable have_partitioning has a value of YES, it indicates MySQL server supports partitioning.

Similarly, you can also verify the partitioning support by issuing the SHOW PLUGINS statement at the mysql prompt, as shown below.

```
mysql> SHOW PLUGINS;
+------------+----------+----------------+---------+---------+
| Name       | Status   | Type           | Library | License |
+------------+----------+----------------+---------+---------+
| binlog     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| partition  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ARCHIVE    | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| BLACKHOLE  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| CSV        | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| FEDERATED  | DISABLED | STORAGE ENGINE | NULL    | GPL     |
| MEMORY     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| InnoDB     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MyISAM     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ndbcluster | DISABLED | STORAGE ENGINE | NULL    | GPL     |
+------------+----------+----------------+---------+---------+
11 rows in set (0.00 sec)
```

If the status is being displayed as ACTIVE against the partition plugin, it indicates the partitioning feature is enabled and available.

Hot Tip To disable the partitioning support option, start the MySQL server with the –skip – partition option. The value of have partitioning is now DISABLED.

MYSQL DATABASE ENGINE – A PREREQUISITE FOR PARTITIONING

For creating partitioned tables, you can use most of the storage engines that are supported by the MySQL server.

Hot Tip It is important to note that all partitions of the partitioned table should be using the same storage engine.

To determine which storage engine the MySQL server supports, issue the SHOW ENGINES statement at the mysql prompt at shown below.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary
tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: MyISAM
     Support: DEFAULT
     Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: InnoDB
     Support: YES
     Comment: Supports transactions, row-level locking, and foreign
keys
Transactions: YES
          XA: YES
```

Among all, the value against support column indicates whether an engine can be used or not.

Hot Tip A value of YES, NO and DEFAULT with the storage engine indicates if an engine is available, not available or available and currently set as default storage, respectively.

PARTITIONING TYPES

The following section outlines the various types of partitioning techniques that are supported by MySQL.

RANGE Partitioning

Features
1.Tables are partitioned based on the column values falling within a given range.
2. Ranges should be contiguous but not overlapping.

The following example illustrates the range-partitioning technique. First, we create an employees table, as shown below:

```
CREATE TABLE employees (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT NOT NULL,
store_id INT NOT NULL
);
```

Among the list of columns, store_id can be used to partition the entire table based on the values available with it, as shown below:

```
CREATE TABLE employees (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT NOT NULL,
store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

It is important to note that each partition in defined in order from lowest to highest.

Hot Tip For accommodating rows with some higher and/or greater values in the partitions, a clause named VALUE LESS THAN is used in the create table.. statement, as show below.

```
CREATE TABLE employees (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT NOT NULL,
store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN
MAXVALUE
);
```

MAXVALUE represents an integer value that is always greater than the largest possible value that is available.

LIST Partitioning

Features
1. Very analogous to range partitioning.
2. Partition is selected based on columns matching a set of discrete values.

Features
3. Each partition should be explicitly defined.
4. Partitions do not need to be declared in any order.

The following example demonstrates how to list partition a table based on a column value.

```
CREATE TABLE employees (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT
)
PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

Hot Tip Deleting data from a partition works more efficiently than deleting the data directly from the table.

HASH Partitioning

Features
1. Ensures an even distribution of data among partitions.
2. Explicitly specify into which partition a given column value is to be stored

The following example demonstrates how to hash partition a table based on a column.

```
CREATE TABLE employees (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

If you do not include the PARTITIONS clause, the number of partitions defaults to 1.

KEY Partitioning

Features
1. Conceptually and syntactically, it is analogous to HASH partitioning.
2. The requisite hashing function is supplied by MySQL server.
3. Partitioning key must consist of table's primary key.

```
CREATE TABLE k1 (
id INT NOT NULL PRIMARY KEY,
name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 2;
```

Hot Tip If there is no primary key available with the table but there is a unique key, then the unique key can be used as a partitioning key, as shown below.

```
CREATE TABLE k1 (
id INT NOT NULL,
name VARCHAR(20),
UNIQUE KEY (id)
)
PARTITION BY KEY()
PARTITIONS 2;
```

RETRIEVING INFORMATION ABOUT EXISTING PARTITIONS

You can obtain information about existing partitions in a number of ways.

Sr. No	Statement
1	Issue SHOW CREATE TABLE statement to view partition clause used.
2	Issue SHOW TABLE STATUS statement to determine whether table is partitioned.
3	Query the INFORMATION SCHEMA.PARTITIONS table for information about table partitions.
4	Issue statement EXPLAIN PARTITIONS SELECT, as shown in below syntax, Issue statement EXPLAIN PARTITIONS SELECT, as shown in below syntax: EXPLAIN PARTITIONS SELECT * FROM table_name WHERE clause

MANAGING PARTITIONS

Partition management refers to a set of activities that deals with the actions, such as:

- Adding
- Dropping
- Redefining
- Merging or Splitting

You can realize all of these actions by using the ALTER TABLE statement with a partition_option clause.

Hot Tip Only a single PARTITION BY, ADD PARTITION, DROP PARTITION, REORGANIZE PARTITION, or COALESCE PARTITION clause can be used in a given ALTER TABLE statement.

Managing Range and List Partitions - a Quick Glance
1. Managing range and list partitions is very similar.
2. Adding and Dropping of partitions are handled in same way.
3. Dropping an existing partitions is very straightforward.
4. Accomplished by using ALTER TABLE statement with appropriate partition clause.
5. To delete all the data from a partition, use the DROP PARTITION clause. ALTER TABLE table_name DROP PARTITION partition_name;

Managing Range and List Partitions - a Quick Glance
6. To preserve the table definition, use TRUNCATE TABLE statement.
7. To change the partition without losing any data, use the REORGANIZE PARTITION clause. ALTER TABLE table_name REORGANIZE PARTITION [partition_name INTO (partition_definitions)]
8. With range partitions, you can add new partitions ONLY to the high end of the partition list.
9. Adding a new partition in between or before an existing partition will result in error.
10. You cannot add a new LIST PARTITION encompassing any values that are already included in the value list of an existing partition. This will result in an error.
11. The REORGANIZE PARTITION clause may also be used for merging adjacent partitions.
12. The REORGANIZE PARTITION clause cannot be used for changing the table's partitioning type. For example, you cannot change Range partitions to Hash partitions, or vice-versa.

Managing Hash and Key Partitions – a Quick Glance
1. When it comes to making changes to partitioning setup, it is similar with hash and key partitions.
2. You cannot drop hash or key partitions.
3. You can merge hash or key partitions by using an ALTER TABLE statement with the coalesce partition clause, as shown below: ALTER TABLE table_name COALESCE PARTITION 4;
4. The number following the coalesce partition clause refers to the number of partitions to remove from the table
5. Attempting to remove more partitions than the table has will lead to an error.

PARTITIONS MAINTENANCE

You may also carry out a number of table and associated partition maintenance tasks by issuing a few SQL statements using the following options:

SQL Options	Description	Equivalent SQL Statement
REBUILDING PARTITIONS	Used for rebuild ing a partition and for defragmentation.	ALTER TABLE table_name REBUILD PARTITION p0, p1;
OPTIMIZING PARTITIONS	Used to reclaim any unused space and defragment partition data files.	ALTER TABLE table_name OPTIMIZE PARTITION p0, p1;
ANALYZING PARTITIONS	Used to store key distributions about partitions.	ALTER TABLE table_name ANALYZE partition p3;
REPAIRING PARTITIONS	Used for repairing any corrupted partitions.	ALTER TABLE table_name REPAIR partition p0, p1;
CHECKING PARTITIONS	Used for checking errors in partitions	ALTER TABLE table_name CHECK partition p1;

PARTITION PRUNING

Partition pruning is an optimization technique that can be implemented for partitioned tables. The idea behind pruning is relatively simple and it can be described as "Do not scan a partition when there are no matching values." For example, suppose that you have a partitioned table t1 defined by the statement shown below:

```
CREATE TABLE t1 (
fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
    PARTITION p0 VALUES LESS THAN (64),
    PARTITION p1 VALUES LESS THAN (128),
    PARTITION p2 VALUES LESS THAN (192),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Now, consider the case where you wish to obtain results from a query such as:

SELECT fname, lname, region_code, dob FROM t1 WHERE region_code > 125 AND region_code < 130;

It is easy to see that none of the rows which ought to be returned will be in either of the partitions p0 or p3; that is, we need to search only in partitions p1 and p2 to find matching rows. By doing so, we can save time and effort and find matching rows instead of having to scan all partitions in the table. This "cutting away" of unneeded partitions is known as "pruning."

The query optimizer can perform pruning whenever a WHERE condition can be reduced to either of the following two cases:

- partition_column = constant
- partition_column IN (constant1, constant2, constant3,...., constantN)

In the first case, the optimizer simply evaluates the partitioning expression for the value given. Then it determines and scans only the partition containing that value. In many cases, the equal sign can be replaced with other arithmetic comparison operators, including <, >, <= , >= , and <>.

Hot Tip Queries using BETWEEN in the WHERE clause can also take advantage of partition pruning.

In the second case, the optimizer evaluates the partitioning expression for each value in the list, creates a list of matching partitions, and then scans only the partitions in that list. Further pruning can be applied to short ranges, which the optimizer can convert into equivalent lists of values.

Hot Tip Pruning can also be applied for tables partitioned on a DATE or DATETIME column when the partitioning expression uses the YEAR() or TO_DAYS() function.

IMPLEMENTING PARTITION PRUNING TECHNIQUE

Partition pruning can be implemented for all partition types. Though pruning for RANGE partitioning is already explained in the above section, this section illustrates how this technique can be applied to other types of partitions as well. For example, consider a table t3 that is partitioned by LIST, as shown below:

```
CREATE TABLE t3
(
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT     NULL,
    dob DATE NOT NULL
)
PARTITION BY LIST(region_code)
(
    PARTITION r0 VALUES IN (1, 3),
    PARTITION r1 VALUES IN (2, 5, 8),
    PARTITION r2 VALUES IN (4, 9),
    PARTITION r3 VALUES IN (6, 7, 10)
);
```

Table t3 shows that the region_code column is limited to values between 1 and 10 (inclusive). So, when a select query is issued, such as the one below—

SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3;

—the optimizer determines the partitions where values 1, 2 and 3 are found, which should be ro, r1 and skips those that remain (r2 and r3).

Similarly, for the tables that are partitioned by HASH and KEY, pruning is also possible in the cases when the WHERE clause uses a simple " = " relation against a column that is used in the partition expression. For example, consider the created table shown below:

```
CREATE TABLE t4 (
fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY KEY(region_code)
PARTITIONS 8;
```

Any query that compares a column value with a constant can be pruned, as shown in the next line of code:

```
SELECT * FROM t4  WHERE region_code = 7;
```

Importantly, pruning can also be employed for short ranges, because the optimizer can turn such conditions into IN relations. For example, consider the next two queries with reference to the previously defined table t4:

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;
```

```
SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

In both of these cases, the WHERE clause is transformed by the optimizer into WHERE region_code IN (3, 4, 5). This optimization is used only if the range size is smaller than the number of partitions.

Hot Tip

Pruning can be used only on integer columns of tables partitioned by HASH or KEY. But, for a table that is partitioned by KEY, has a composite primary key, and uses a composite partitioning key, it is possible to perform pruning for queries meeting the following two criteria:

- The query must have a WHERE clause of the form pkcol1 = c1 AND pkcol2 = c2 AND ... pkcolN = cN, where pkcol1..... pkcolN are the partitioning key columns and c1.....cN are constant values.
- All columns of the partitioning key must be referenced in the WHERE clause.

ABOUT 'EXPLAIN PARTITION' STATEMENT

1. Generally, EXPLAIN statement is used to obtain information about how MySQL executes a statement.
2. EXPLAIN PARTITION is highly helpful for examining queries involving partitions.
3. A SELECT statement that is preceded by an EXPLAIN statement displays information from the Optimizer about the query execution plan.
4. For Example: EXPLAIN PARTITIONS SELECT select_options

RESTRICTIONS AND LIMITATIONS ON PARTITIONING

1. Use of the arithmetic operators +, -, and * is permitted in partitioning expressions. The result must be an integer value or NULL (except in the case of [LINEAR] KEY partitioning).
2. The bit operators |, &, ^, <<, >>, and ~ are not permitted in partitioning expressions.
3. Tables employing user-defined partitioning do not preserve the SQL mode in effect at the time that they were created.
4. A change in the SQL mode at any time after the creation of partitioned tables may lead to major changes in the behavior of such tables, and could easily lead to corruption or loss of data.
5. Sometimes a change in the server SQL mode can make partitioned tables unusable.
6. Server SQL modes also impact replication of partitioned tables since differing SQL modes on master and slave can lead to differently evaluated partitioning expressions.
7. Different SQL modes on master and slave can cause different distributions of data among partitions.
8. Partitioned tables do not support FULLTEXT indexes or searches. This includes MyISAM storage engine.
9. Columns with spatial data types, such as POINT or GEOMETRY, cannot be used in partitioned tables.
10. Temporary tables cannot be partitioned.
11. A partitioning key must be either an integer column or an expression that resolves to an integer. The column or expression value may also be NULL.
12. Partitioned tables do not support foreign keys.

PERFORMANCE CONSIDERATIONS FOR PARTITIONING

1. Generally, partitioning and repartitioning operations depend on file-system operations for their implementation.
2. Speed of the partitioning-related operations is affected by factors such as file system type, disk speed, swap space and file-handling efficiency.
3. In particular, you should make sure that large_files_support is enabled and that open_files_limit is set properly.
4. Partitioned tables using the MyISAM storage engine, therefore increasing myisam_max_sort_file_size, may improve performance.
5. Similarly, partitioning and repartitioning operations involving InnoDB tables may be made more efficient by enabling innodb_file_per_table.
6. Partitioning operations, queries, and update operations generally tend to be faster with MyISAM tables than with InnoDB or NDB tables.
7. As with non-partitioned tables, proper use of indexes can significantly speed up queries on partitioned tables.
8. The maximum possible number of partitions for a given table (that does not use the NDB storage engine) is 1024. This includes sub-partitions.

CONCLUSIONS

Database partitioning is a value-added database design technique that many data modelers and DBAs rely on to achieve various objectives.

a) Reduces the amount of data read operations for typical SQL operations. As such, the overall response time is reduced.

b) Increases database performance by optimizing the database scan operations internally.

c) Simplifies data management so there is more control over how the data is organized inside a database.

d) Achieves a greater degree of query throughput by spreading data more logically.

SUGGESTED READINGS

1. MySQL Reference Manual, which is available through the website dev.mysql.com/doc/en/

2. MySQL Cookbook, by Paul DuBois

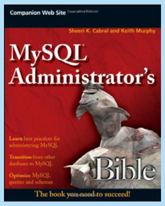
3. MySQL in a Nutshell, by Russell Dyer

4. MySQL Administrators Bible, by Sheeri K. Cabral and Keith Murphy.

ABOUT THE AUTHOR

Narayana Maruvada is a computer science and engineering graduate, currently working as a QA engineer at Benefitfocus, the country's leading provider of benefits technology. He is part of a top notch technology organization that focuses on rapid, high quality, and superiorly architected solutions. Narayana has over 6 years of experience developing and testing web-based applications. In particular, Narayana has worked extensively with MySQL and Oracle databases, tuning queries and configurations for maximum performance and reliability.

RECOMMENDED BOOK


Companion Web Site
Sheeri K. Cabral and Keith Murphy
MySQL Administrator's Bible
The book you need to succeed!

This book provides a solid framework for both database novices and experienced DBA's transitioning to MySQL. It provides essential coverage of the fundamentals of MySQL database management, including MySQL's unique approach to basic database features and functions as well as coverage of SQL queries, data and index types, stored-procedure, functions, triggers, views, transactions, etc. It also offers comprehensive coverage of advanced topics such as MySQL server tuning, managing storage engines, caching, backup and recovery, managing users, index tuning, database and performance monitoring, security, and more.

ISBN-13: 978-1-936502-51-6
ISBN-10: 1-936502-51-8
50795
9 781936 502516
$7.95

Version 1.0