18.1. Overview of Partitioning in MySQL

This section provides a conceptual overview of partitioning in MySQL 5.1.

For information on partitioning restrictions and feature limitations, see Section 18.5, “Restrictions and Limitations on Partitioning”.

The SQL standard does not provide much in the way of guidance regarding the physical aspects of data storage. The SQL language itself is intended to work independently of any data structures or media underlying the schemas, tables, rows, or columns with which it works. Nonetheless, most advanced database management systems have evolved some means of determining the physical location to be used for storing specific pieces of data in terms of the file system, hardware or even both. In MySQL, the **InnoDB** storage engine has long supported the notion of a tablespace, and the MySQL Server, even prior to the introduction of partitioning, could be configured to employ different physical directories for storing different databases (see Section 8.9.6, “Using Symbolic Links”, for an explanation of how this is done).

Partitioning takes this notion a step further, by enabling you to distribute portions of individual tables across a file system according to rules which you can set largely as needed. In effect, different portions of a table are stored as separate tables in different locations. The user-selected rule by which the division of data is accomplished is known as a partitioning function, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function. The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression. This expression can be a column value, a function acting on one or more column values, or a set of one or more column values, depending on the type of partitioning that is used.

In the case of **RANGE**, **LIST**, and [**LINEAR**] **HASH** partitioning, the value of the partitioning column is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored. This function must be nonconstant and nonrandom. It may not contain any queries, but may use an SQL expression that is valid in MySQL, as long as that expression returns either **NULL** or an integer ***intval*** such that

```
-MAXVALUE <= intval <= MAXVALUE
```

(**MAXVALUE** is used to represent the least upper bound for the type of integer in question. **-MAXVALUE** represents the greatest lower bound.)

For [**LINEAR**] **KEY**, the partitioning expression consists of a list of one or more columns, and the partitioning function is supplied by MySQL.

For more information about permitted partitioning column types and partitioning functions, see Section 18.2, “Partitioning Types”, as well as Section 13.1.17, “**CREATE TABLE** Syntax”, which provides partitioning syntax descriptions and additional examples. For information about restrictions on partitioning functions, see Section 18.5.3, “Partitioning Limitations Relating to Functions”.

This is known as horizontal partitioning—that is, different rows of a table may be assigned to different physical partitions. MySQL 5.1 does not support vertical partitioning, in which different columns of a table

are assigned to different physical partitions. There are not at this time any plans to introduce vertical partitioning into MySQL 5.1.

For information about determining whether your MySQL Server binary supports user-defined partitioning, see Chapter 18, *Partitioning*.

For creating partitioned tables, you can use most storage engines that are supported by your MySQL server; the MySQL partitioning engine runs in a separate layer and can interact with any of these. In MySQL 5.1, all partitions of the same partitioned table must use the same storage engine; for example, you cannot use **MyISAM** for one partition and **InnoDB** for another. However, there is nothing preventing you from using different storage engines for different partitioned tables on the same MySQL server or even in the same database.

MySQL partitioning cannot be used with the **MERGE** or **CSV** storage engines. Beginning with MySQL 5.1.15, **FEDERATED** tables also cannot be partitioned (Bug #22451). Prior to MySQL 5.1.6, it was also not feasible to create a partitioned table using the **BLACKHOLE** storage engine (Bug #14524). See Section 18.5.2, “Partitioning Limitations Relating to Storage Engines”.

Partitioning by **KEY** or **LINEAR KEY** is possible with **NDBCLUSTER**, but other types of user-defined partitioning are not supported for tables using this storage engine. In addition, an **NDBCLUSTER** table that employs user-defined partitioning must have an explicit primary key, and any columns referenced in the table's partitioning expression must be part of the primary key. However, if no columns are listed in the **PARTITION BY KEY** or **PARTITION BY LINEAR KEY** clause of the **CREATE TABLE** or **ALTER TABLE** statement used to create or modify a user-partitioned **NDBCLUSTER** table, then the table is not required to have an explicit primary key. For more information, see Section 17.1.6.1, “Noncompliance with SQL Syntax in MySQL Cluster”.

To employ a particular storage engine for a partitioned table, it is necessary only to use the **[STORAGE] ENGINE** option just as you would for a nonpartitioned table. However, you should keep in mind that **[STORAGE] ENGINE** (and other table options) need to be listed *before* any partitioning options are used in a **CREATE TABLE** statement. This example shows how to create a table that is partitioned by hash into 6 partitions and which uses the **InnoDB** storage engine:

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
    ENGINE=INNODB
    PARTITION BY HASH( MONTH(tr_date) )
    PARTITIONS 6;
```

Each **PARTITION** clause can include a **[STORAGE] ENGINE** option, but in MySQL 5.1 this has no effect.

> **Important**
>
> Partitioning applies to all data and indexes of a table; you cannot partition only the data and not the indexes, or *vice versa*, nor can you partition only a portion of the table.

Data and indexes for each partition can be assigned to a specific directory using the **DATA DIRECTORY** and **INDEX DIRECTORY** options for the **PARTITION** clause of the **CREATE TABLE** statement used to create the partitioned table.

Prior to MySQL 5.1.18, these options were permitted even when the **NO_DIR_IN_CREATE** server SQL mode was in effect (Bug #24633).

The **DATA DIRECTORY** and **INDEX DIRECTORY** options have no effect when defining partitions for tables using the **InnoDB** storage engine.

`DATA DIRECTORY` and `INDEX DIRECTORY` are not supported for individual partitions or subpartitions on Windows. Beginning with MySQL 5.1.24, these options are ignored on Windows, except that a warning is generated (Bug #30459)-

In addition, `MAX_ROWS` and `MIN_ROWS` can be used to determine the maximum and minimum numbers of rows, respectively, that can be stored in each partition. The `MAX_ROWS` option can be useful for causing MySQL Cluster tables to be created with extra partitions, thus allowing for greater storage of hash indexes. See the documentation for the `DataMemory` data node configuration parameter, as well as Section 17.1.2, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”, for more information.

Some advantages of partitioning are listed here:

- Partitioning makes it possible to store more data in one table than can be held on a single disk or file system partition.
- Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding one or more new partitions for storing specifically that data.
- Some queries can be greatly optimized in virtue of the fact that data satisfying a given `WHERE` clause can be stored only on one or more partitions, which automatically excludes any remaining partitions from the search. Because partitions can be altered after a partitioned table has been created, you can reorganize your data to enhance frequent queries that may not have been often used when the partitioning scheme was first set up. This ability to exclude non-matching partitions (and thus any rows they contain) is often referred to as partition pruning, and was implemented in MySQL 5.1.6. For more information, see Section 18.4, “Partition Pruning”.

Other benefits usually associated with partitioning include those in the following list. These features are not currently implemented in MySQL Partitioning, but are high on our list of priorities.

- Queries involving aggregate functions such as `SUM()` and `COUNT()` can easily be parallelized. A simple example of such a query might be `SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id;`. By “parallelized,” we mean that the query can be run simultaneously on each partition, and the final result obtained merely by summing the results obtained for all partitions.
- Achieving greater query throughput in virtue of spreading data seeks over multiple disks.

Be sure to check this section and chapter frequently for updates as MySQL Partitioning development continues.

Previous / Next / Up / Table of Contents

User Comments

Add your own comment.

Top / Previous / Next / Up / Table of Contents