Using MySQL 5.6 (and above) with Policy Manager 8.0 and Community Manager 8.0

Using MySQL 5.6 (and above) with Policy Manager 8.0 and Community Manager 8.0

using_mysql_56_pm80_cm80
January 2016

Copyright

Trademarks

Akana, Inc.

Akana, Inc.
12100 Wilshire Blvd, Suite 1800
Los Angeles, CA 90025
(866) SOA-9876
www.akana.com
info@akana.com

Disclaimer

Introduction

If you are using MySQL 5.6 or higher with Policy Manager 8.0 and/or Policy Manager 8.0 and Community Manager 8.0, the following MySQL property settings must be configured as outlined in the following sections.

MySQL Properties

innodb_file_per_table

When `innodb_file_per_table` is enabled (the default in MySQL 5.6.6 and higher), `InnoDB` stores the data and indexes for each newly created table in a separate `.ibd` file, rather than in the system tablespace. The storage for these `InnoDB` tables is reclaimed when the tables are dropped or truncated.

`innodb_file_per_table` is dynamic and can be set `ON` or `OFF` using `SET GLOBAL`. You can also set this parameter in the MySQL configuration file (`my.cnf` or `my.ini`) but this requires shutting down and restarting the server.

Akana requires this setting enabled so tables can use DYNAMIC row format.

```
set global innodb_file_per_table=ON;
```

Advantages

- Tables created in file-per-table tablespaces use the Barracuda file format. The Barracuda file format enables features such as compressed and dynamic row formats. Tables created in the system tablespace cannot use these features.
- You can reclaim disk space when truncating or dropping a table stored in a file-per-table tablepace. Truncating or dropping tables stored in the system tablespace creates free space internally in the system tablespace data files (ibdata files) which can only be used for new `InnoDB` data.
- The `TRUNCATE TABLE` operation is faster when run on tables stored in file-per-table tablepaces.
- You can run `OPTIMIZE TABLE` to compact or recreate a file-per-table tablespace. When you run an `OPTIMIZE TABLE`, `InnoDB` creates a new `.ibd` file with a temporary name, using only the space required to store actual data. When the optimization is complete, `InnoDB` removes the old `.ibd` file and replaces it with the new one. If the previous `.ibd` file grew significantly but the actual data only accounted for a portion of its size, running `OPTIMIZE TABLE` can reclaim the unused space.
- File-per-table tablespaces may improve chances for a successful recovery and save time when a corruption occurs, when a server cannot be restarted, or when backup and binary logs are unavailable.
- You can back up or restore individual tables quickly using the MySQL Enterprise Backup product, without interrupting the use of other `InnoDB` tables. This is beneficial if you have tables that require backup less frequently or on a different backup schedule.
- File-per-table tablespaces are convenient for per-table status reporting when copying or backing up tables.
- You can monitor table size at a file system level, without accessing MySQL.

- The system tablespace stores the data dictionary and undo logs, and has a 64TB size limit. By comparison, each file-per-table tablespace has a 64TB size limit, which provides you with room for growth.

Disadvantages

- With file-per-table tablespaces, each table may have unused space, which can only be utilized by rows of the same table. This could lead to wasted space if not properly managed.
- `fsync` operations must run on each open table rather than on a single file. Because there is a separate `fsync` operation for each file, write operations on multiple tables cannot be combined into a single I/O operation. This may require `InnoDB` to perform a higher total number of `fsync` operations.
- **mysqld** must keep one open file handle per table, which may impact performance if you have numerous tables in file-per-table tablespaces.
- If many tables are growing there is potential for more fragmentation which can impede `DROP TABLE` and table scan performance. However, when fragmentation is managed, having files in their own tablespace can improve performance.

innodb_large_prefix

By default, an index key for a single-column index can be up to 767 bytes. When this option is enabled, index key prefixes longer than 767 bytes (up to 3072 bytes) are allowed for `InnoDB` tables that use the `DYNAMIC` and `COMPRESSED` row formats.

Akana requires this setting enabled since it uses index keys that are greater than 767 bytes in length.

```
set global innodb_large_prefix=ON;
```

innodb_file_format

The file format to use for new `InnoDB` tables. Currently, `Antelope` and `Barracuda` are supported. This setting only applies to tables that have their own file-per-table tablespace, so for it to have an effect, `innodb_file_per_table` must be enabled. The Barracuda file format is required to use Compressed or Dynamic row formats and associated features such as compression, off-page storage for large variable-length columns, and large index key prefixes.

Akana requires that the file format be set to Barracuda so tables can be created with DYNAMIC row format.

```
set global innodb_file_format=barracuda;
```

Advantages

- The `DYNAMIC` row format maintains the efficiency of storing long data values such as BLOB and TEXT.

max_allowed_packet

The maximum size of one packet or any generated/intermediate string, or any parameter sent by the application to the server. The default is 4MB.

You must increase this value if you are using large BLOB columns or long strings. It should be as big as the largest BLOB you want to use. The protocol limit for `max_allowed_packet` is 1GB. The value should be a multiple of 1024; non-multiples are rounded down to the nearest multiple.

Akana requires the packet size set to `1073741824` (1GB) since BLOB and TEXT columns are used.

```
set global max_allowed_packet=1073741824;
```

innodb_flush_log_at_trx_commit

Controls the balance between strict ACID compliance for commit operations, and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. You can achieve better performance by changing the default value, but then you can lose up to a second of transactions in a crash.

- The default value of 1 is required for full ACID compliance. With this value, the contents of the InnoDB log buffer are written out to the log file at each transaction commit and the log file is flushed to disk.
- With a value of 0, the contents of the InnoDB log buffer are written to the log file approximately once per second and the log file is flushed to disk. No writes from the log buffer to the log file are performed at transaction commit. Once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues. Because the flush to disk operation only occurs approximately once per second, you can lose up to a second of transactions with any **mysqld** process crash.
- With a value of 2, the contents of the InnoDB log buffer are written to the log file after each transaction commit and the log file is flushed to disk approximately once per second. Once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues. Because the flush to disk operation only occurs approximately once per second, you can lose up to a second of transactions in an operating system crash or a power outage.

Akana does not require this setting changed from default value ‘1’. However if you are seriously looking at improving the performance, the value can be changed to ‘0’ or ‘2’.

```
set global innodb_flush_log_at_trx_commit=0;
```