UP.TIME DATA MANAGEMENT PACK:

MYSQL TABLE PARTITIONING

TABLE OF CONTENTS

TABLE OF CONTENTS **2**

REVISION HISTORY **4**

INTRODUCTION **4**

About up.time 4
Limitations of Archiving on Very Large Databases (VLDBs) 4
Issues Caused by Archiving on Very Large Databases (VLDBs) 4
The Solution: MySQL Table Partitioning 4
Selecting the appropriate Partition Type 5

TECHNICAL INFORMATION **5**

Requirements 5
Limitations 5
Table Partition Types Used 5

OVERVIEW OF CONVERSION STEPS **7**

PRE-CONVERSION STEPS **8**

Verify Necessary Free Space 8
Backup the Database 8
Disable Built-In Archiving 8
Stop up.time Services 9
Clear Remaining Archive Data 9
Disable Default up.time Datastore (MySQL) 9
Upgrade to MySQL 5.1 10

CONVERSION STEPS **13**

How to Execute the Scripts 13
Create New Partitioned Tables 13
Copy Data into New Partitioned Tables 13
Add Database Triggers 14
Start up.time 15

MAINTENANCE STEPS **16**

Add Automated Maintenance Procedures 16
Monitor MySQL Event Scheduler 16

MANAGING THE PARTITIONS **17**

INFORMATION 17
RUNNING CHECKS ON THE PARTITIONS 17

REVISION HISTORY

Rev	Date	Author	Description
1	Nov. 12, 2010	Joel Pereira	Initial Publication
2	Jan. 17, 2011	Joel Pereira	Clarification on a few sections
3	Mar. 2, 2011	Joel Pereira	Fixed performance tables to be views (issue when graphing)
4	Apr. 26, 2011	Joel Pereira	Reworded confusing steps during MySQL setup

INTRODUCTION

About up.time

up.time is an On-Line Transaction Processing (OLTP) application; meaning it inserts a huge amount of (small) transactions into the database at a constant and high rate. This can be an intensive process on a database as it is constantly writing new data.

After adding many systems to up.time and having it collect performance data for a lengthy period of time there will be a point where we are no longer interested in data beyond a certain point in time (example: anything older than 6 months, or 1 year, etc). This is where built-in archiving is used to remove data older than the specified amount of time (number of months).

Limitations of Archiving on Very Large Databases (VLDBs)

up.time archiving works by first extracting any old data (older than the archiving policy set) from the database, dumping the data into archive files (zipped XML files, "*.xml.gz") and then going back and removing all the archived data out of the database.

The problem is that this is a very intensive process on the tables that are also being constantly written to with new performance data. This can cause major issues having 2 intense processes both fighting to work on the same large tables in the database.

Issues Caused by Archiving on Very Large Databases (VLDBs)

For most users, the archiving process should work without a problem, but for users with a larger setup (monitoring over 500 systems on one monitoring station) they may experience some of the issues below:

- Performance degradation
- MySQL process utilizing high CPU resources (during archiving)
- Archiving process failing to complete successfully

The Solution: MySQL Table Partitioning

To get around the limitation of having to manage the huge amounts of data directly, table partitioning allows us to eliminate the archiving process entirely by segmenting the large tables that hold performance data into smaller "partitions". Each “partition” holds one month of data, which allows us to easily and quickly manage older data by swapping it out of the database without having to deal with long wait times and performance degradation of the built-in archiving method.

Selecting the appropriate Partition Type

We will be using Range partitioned tables in MySQL as it is the most suitable partitioning type for these tables.

Unfortunately, some of the tables will require us to add a new column containing a date. This extra column is necessary on all of the "performance_*" tables, except for performance_sample, in the up.time schema. This is a limitation in MySQL in that there is no other method of using partitioning on those tables without doing so. We will also have to create triggers to manage the extra columns.

Link: http://dev.mysql.com/doc/refman/5.1/en/partitioning.html

TECHNICAL INFORMATION

Requirements

- MySQL 5.1 Community Server
  - As of writing this document the latest version is MySQL 5.1.52 but there should not be a problem with newer 5.1 builds

Limitations

Table Partition Types Used

The table below indicates the table partitioning type used for each table in the up.time schema that requires partitioning.

Original Table Name	Modified Table Name (if empty, using original name)	Partitioning Column	Type of Partitioning
performance_sample		sample_time	Range Partitioning
performance_aggregate	performance_aggregate_par	sample_time	Range Partitioning
performance_cpu	performance_cpu_par	sample_time	Range Partitioning
performance_disk	performance_disk_par	sample_time	Range Partitioning
performance_disk_total	performance_disk_total_par	sample_time	Range Partitioning
performance_esx3_workload	performance_esx3_workload_par	sample_time	Range Partitioning
performance_fscap	performance_fscap_par	sample_time	Range Partitioning
performance_lpar_workload	performance_lpar_workload_par	sample_time	Range Partitioning
performance_network	performance_network_par	sample_time	Range Partitioning
performance_nrm	performance_nrm_par	sample_time	Range Partitioning
performance_psinfo	performance_psinfo_par	sample_time	Range Partitioning
performance_vxvol	performance_vxvol_par	sample_time	Range Partitioning
performance_who	performance_who_par	sample_time	Range Partitioning
erdc_int_data		sampletime	Range Partitioning
erdc_decimal_data		sampletime	Range Partitioning
erdc_string_data		sampletime	Range Partitioning
ranged_object_value		sample_time	Range Partitioning

We will recreate the tables listed above with partitions and then perform mass “insert into…select” queries to copy the data from the original tables into the new ones before renaming the partitioned tables to the original name. We will also create triggers for the performance_* tables (except performance_sample).
After the process is complete and all the tables have been converted to partitioned tables, everything works the exact same way as before. The biggest thing that is greatly enhanced is the fact that we’ll be able to make modifications (move/drop) to monthly chunks of data without experiencing any performance issues on either the database or up.time side. Managing the partitioned tables is seamless to up.time and should not impact anything being done on the application side, so changes can be done while the application is online.

Once we verify that the application and database is working fully as expected we will delete the old tables and add the automated maintenance procedures so MySQL can auto-manage the partitions without any assistance or intervention. These automations are automatically triggered via the built-in MySQL Event Scheduler. We will create a monitor in up.time to verify this process is always running.

OVERVIEW OF CONVERSION STEPS

To convert the current MySQL database to use partitioned tables we must first configure and install MySQL 5.1 and then perform the conversion on all performance tables to be partitioned tables. Once that is complete and operational we can add the fully automated maintenance scripts to that the database will auto-manage the partitions.

The following is the list of steps we will take from start to finish for converting to partitioned tables.

Pre-Conversion Steps

1. Verify Necessary Free Space
2. Backup the Database
3. Disable Built-In Archiving
4. Stop up.time Services
5. Clear Remaining Archive Data
6. Disabling Default up.time Datastore (MySQL)
7. Upgrade to MySQL 5.1

Conversion Steps

1. Create New Partitioned Tables
2. Copy Data into New Partitioned Tables
3. Add Database Triggers
4. Start up.time

Maintenance Steps

1. Add Automated Maintenance Procedures
2. Monitor MySQL Event Scheduler

PRE-CONVERSION STEPS

Verify Necessary Free Space

The most important thing to do before proceeding to the conversion steps is to make sure that we have enough free space to perform all the conversion(s).

The scripts will create newly partitioned tables and then copy all the data from the original tables into the new ones, so it's important that there is enough free space in the database for this to work properly. It will run through this process one table at a time so it should be sufficient to have enough free space for a copy of the largest performance table in the database plus a little bit extra. Generally if you have enough free space for a copy of the largest table in your database times two it should be safe.

If you aren't sure or have any doubts about sizing issues, feel free to contact uptime support (support@uptimesoftware.com) with any questions.

Backup the Database

Important Note: Before we proceed to start converting the up.time performance tables to partitioned tables, we HIGHLY recommend taking a full backup of the database. The next few steps will perform large changes to (potentially) very large tables in the database, which means that unexpected events can happen that may corrupt the database. There may also be certain limitations or restrictions that cause issues while running the conversion that may result in corrupted data. If anything unexpected should happen, we may rely on the database backup to be able to revert back to the previous point. If there is no database backup then it is possible that all historical performance data may be lost if there is an issue.

Disable Built-In Archiving

Since we will be enabling Table Partitioning to remove the older data, we will have to disable built-in archiving so it does not interfere with partitioning.

Archiving can be disabled by:

- Login to the up.time interface as a "superadmin" user
- Click on the Config tab and click on the "Archive Policy" link on the left
- Uncheck the "Enable Archiving" checkbox
- Click on the "Set Archive Policy" button

Stop up.time Services

Before running any scripts we should stop all the up.time instances so they do not cause any issues. Any and all up.time data collectors, UI, reporting instances that access the database should be stopped.

How to stop up.time on Windows:

- Start > Run > services.msc
- Locate and stop the service “up.time Data Collector”
- Locate and stop the service “up.time DataStore”

How to stop up.time on Linux/Solaris:

```
# /etc/init.d/uptime_core stop
# /etc/init.d/uptime_datastore stop
```

Clear Remaining Archive Data

There are two types of archiving data that can be cleaned/deleted.

1. Archived Files - These are files that contain older performance data that can later be re-imported
2. Temporary Archive Information in the Database - This is meta-data used to describe which data has already been archived so that it can later be deleted by another process. This can be freely deleted as it will just be taking up space in the database and not used for partitioning. The table name is “archive_delenda”.

Archived Files

Archived files are stored in “<uptime_dir>/archive” directory.
These can be moved/copied/deleted as they are filled with data that was extracted from the database. They are no longer linked to anything.

Temporary Archive Information in the Database

We can clean a specific table (archive_delenda) in the database that only contains meta-data (not actual data) during the archiving process. This can be deleted without any impact to any of the data.

- Make sure the up.time data collector/core is stopped
- Login to the MySQL database
- Run the following query to clear the table:
    - Truncate table archive_delenda;

That’s it.

Disable Default up.time Datastore (MySQL)

Since we’ll be using a newer version of MySQL, we must disable the currently built-in up.time database (MySQL 5.0) from starting so that it does not cause any conflicts. This involves disabling the service from startup as well as disabling future upgrades to start it up as well.

Windows Monitoring Stations

- Set the “up.time DatStore” service to have a startup type of “Disabled” and stop the service if it is still running

Linux/Solaris Monitoring Stations

- Remove the following file(s):

```
# rm -f /etc/rc.d/rc3.d/S99uptime_datastore
# rm -f /etc/rc.d/rc3.d/K99uptime_datastore
# rm -f /etc/rc.d/rc5.d/S99uptime_datastore
# rm -f /etc/rc.d/rc5.d/K99uptime_datastore
# rm -f /etc/init.d/uptime_datastore
```

Upgrade to MySQL 5.1

This step actually consists of a few short steps:

1. Downloading/installing MySQL 5.1
2. Configuring new MySQL to use uptime database
3. Remove old InnoDB log files
4. Start MySQL 5.1 process
5. Upgrade database tables for MySQL 5.1

Downloading/installing MySQL 5.1

Download the latest MySQL Community Server (MySQL Server) 5.1 (MSI package or RPM package). You can select the appropriate architecture (x86/32bit/64bit) for the system it's being installed on; the different architectures will not affect uptime core interoperability so long as the MySQL database can be started and runs properly.

We need to configure the MySQL 5.1 server for an OLTP (transactional) type database. For more info please contact uptime support (support@uptimesoftware.com).

Configuring new MySQL to use uptime database

Warning: Before proceeding any further, make sure you have a full backup of the up.time database as the next steps will make modifications to the database that cannot be undone. Without a full backup, it is possible to lose all data in the uptime database!

Once MySQL 5.1 is installed, we need to configure it to point to the uptime data files.

- Edit the my.ini file in the MySQL directory and modify the following lines:

```
port=3306
to:
port=3308
```

And add the following line just under the "port" setting:

```
event_scheduler=ON
```

And modify:

```
datadir=”..."
to:
datadir="C:/Program Files/uptime software/uptime/datastore/data/"
```

Make sure to use the correct directory to where the datastore files are located if they are not in the default directory.

Note: Always make sure to always use forward slashes (/) for the path regardless of operating system. This is the format MySQL requires.

Remove old InnoDB log files

NOTE: One critical note here before we proceed is that before deleting the InnoDB log files we must verify there are no errors or corruption issues in the current database. The easiest way to do this is to start and stop the original uptime MySQL service (up.time DataStore or /usr/local/uptime/scripts-*/uptime_datastore) and verify there are no errors in the MySQL error log file. The MySQL error log is located in the <uptime_dir>/datastore/data directory and it will be named “<hostname>.err” (where <hostname> is the hostname of the server).

Example of MySQL error log with no errors:

```
101111 12:17:40 [Note]
101111 12:37:27  InnoDB: Started; log sequence number 0 1387606
101111 12:37:27 [Note] C:\uptime\mysql\bin\mysqld-nt.exe: ready for connections.
Version: '5.0.67-community-nt'  socket: ''  port: 3308  MySQL Community Edition
(GPL)
101111 12:37:51 [Note] C:\uptime\mysql\bin\mysqld-nt.exe: Normal shutdown

101111 12:37:51  InnoDB: Starting shutdown...
101111 12:37:53  InnoDB: Shutdown completed; log sequence number 0 1387606
101111 12:37:53 [Note] C:\uptime\mysql\bin\mysqld-nt.exe: Shutdown complete
```

In the above example output you can see the MySQL server start up, and the shutdown command 30 seconds later.

Due to changes in MySQL 5.1, the old InnoDB log files (ib_logfile0 and ib_logfile1) may not work in the newer version and prevent MySQL from starting up at all. Because of this we need to delete the files so MySQL 5.1 can start successfully. The files will be re-created during start up.

Once you verify there are no errors in the MySQL error log file, stop any MySQL that is still running and simply delete the following two files from the uptime datastore directory:

- <uptime_dir>/datastore/data/ib_logfile0
- <uptime_dir>/datastore/data/ib_logfile1

Start MySQL 5.1 process

Once all of the above changes and checks have been completed we can now start MySQL 5.1. Simply start the new MySQL service or process.

If it starts up without any errors we can proceed.
If there are any issues during startup, review all the steps above, and if issue persists, contact uptime support (support@uptimesoftware.com) before proceeding.

Upgrade database tables for MySQL 5.1

Before proceeding we must upgrade the database tables so they are compatible with MySQL 5.1 features. To do this we simply need to execute the following command(s).

```
> cd <mysql_dir>/bin
> mysql_upgrade.exe -uroot -puptimerocks -P3308 --protocol=tcp
```

You should see output like the following with a successful message at the end:

```
Looking for 'mysql.exe' in: C:\uptime\mysql\bin\mysql.exe
Looking for 'mysqlcheck.exe' in: C:\uptime\mysql\bin\mysqlcheck.exe
Running 'mysqlcheck'...
mysql.columns_priv                                OK
mysql.db                                          OK
mysql.func                                        OK
...
uptime.value_counter                              OK
uptime.vmo_action                                 OK
uptime.vmo_action_input                           OK
Running 'mysql_fix_privilege_tables'...
OK
```

If there are no errors we can proceed.

CONVERSION STEPS

Now that we have upgraded to MySQL 5.1, which supports table partitioning, we can create and transfer all data into newly partitioned tables for a much more efficient data management and archival process.

How to Execute the Scripts

The scripts that will perform the conversion to partitioned tables are meant to be executed via the MySQL client command line tool. There are lots of GUI tools that can be used to execute the scripts as well.

To run a script:

- From the command line, change directory to the "mysql/bin" directory and run the following command to open the MySQL client:
  > mysql -uroot -puptimerocks -P3308 --protocol=tcp uptime

Note: If your uptime deployment was upgraded from uptime 4, the database name "uptime" may not work. If this is the case, edit the file "<uptime_dir>/uptime.conf" and use the database name specified on the line that starts with "dbName=".

Create New Partitioned Tables

To convert the tables to partitioned tables we will need to create copies of the performance tables, but the new tables are already partitioned. We will also have to make some adjustments/changes to some of the performance tables so that they all have a datetime field.

To create the new partitioned tables:

- Access the following link to generate updated "create table" statements with partition information:
  Link: http://support.uptimesoftware.com/partitioning-mysql-creates.php
- You can select the number of partitions (months of data to keep) from the drop-down list.
- Copy & paste the generated SQL commands into the MySQL prompt and execute the commands by hitting Enter on your keyboard
- If it ran successfully you should see the following output:

```
...
Query OK, #### rows affected (0.23 sec)
...
Query OK, #### rows affected (0.23 sec)
```

If there are any errors in the output please contact uptime support (support@uptimesoftware.com) with the error(s) provided and we will be glad to help you resolve the issue.

Copy Data into New Partitioned Tables

This next step will copy all the data in the original performance tables into the newly created partitioned tables. It will then drop the old tables, and rename the new ones to the same names as the originals. It will perform these entire steps one table at a time.

Example:

- It will copy all the data from the "performance_sample" table into the "performance_sample_par" table
- After the copy is done, it will rename the "performance_sample" table to "performance_sample_ori" and the "performance_sample_par" to "performance_sample"
- Once renaming is done, it will drop the "performance_sample_ori" table (actually the performance_sample_ori table is a bit special in that it has dependant tables, so we actually only delete the performance_sample_ori table after the dependant tables have been deleted)

Note: It is important to note that this process may take a lengthy period of time depending on how much data is in the database! We cannot estimate how long this may take as it completely depends on how much data is in each table, server performance, disk performance, etc. We recommend testing this on a test environment with a copy of all your data to get a baseline of how long it will take. After you have a baseline you can schedule a maintenance period so your users are not impacted by the changes.

Note: Some of the tables will instead of "Views" created for them. Views act as a virtual table. Views must be created for the tables that have an extra column added for partitioning. The views have all the same columns and data as the original tables except for one column (sample_time). These tables are listed below.

Table Name	View Name	Hidden Column
performance_aggregate_par	performance_aggregate	sample_time
performance_cpu_par	performance_cpu	sample_time
performance_disk_par	performance_disk	sample_time
performance_disk_total_par	performance_disk_total	sample_time
performance_esx3_workload_par	performance_esx3_workload	sample_time
performance_fscap_par	performance_fscap	sample_time
performance_lpar_workload_par	performance_lpar_workload	sample_time
performance_network_par	performance_network	sample_time
performance_nrm_par	performance_nrm	sample_time
performance_psinfo_par	performance_psinfo	sample_time
performance_vxvol_par	performance_vxvol	sample_time
performance_who_par	performance_who	sample_time

To copy the data from the original tables into the new ones, and to drop/rename the tables:

- Simply execute the script contents of "2_copy_data.sql"
- If it ran successfully you should see many lines of the following output:

```
...
Query OK, #### rows affected (0.23 sec)
...
Query OK, #### rows affected (0.23 sec)
```

If there are any errors in the output please contact uptime support (support@uptimesoftware.com) with the error(s) provided and we will be glad to help you resolve the issue.

Add Database Triggers

Due to the way partitioned tables work in MySQL, all tables must have a datetime column that will be used for partitioning. The problem is that many of the performance tables do not have a date column as they are linked off a main performance table (performance_sample). We have gotten around this by adding date fields to the tables that

did not originally have a date field, but this only solves half the problem; now we have to set the date on those new columns for each table insert. To solve this problem we will use database triggers to have the database take care of setting that column automatically.

To add the necessary triggers to the database:

- Simply execute the script contents of "3_ triggers.sql"
- If it ran successfully you should see the following output:

```
...
Query OK, #### rows affected (0.23 sec)
...
Query OK, #### rows affected (0.23 sec)
```

If there are any errors in the output please contact uptime support (support@uptimesoftware.com) with the error(s) provided and we will be glad to help you resolve the issue.

Start up.time

If we get to this point and all of the above steps executed without any error(s), we should now be able to start up.time and it should run without a problem on the newly partitioned tables. As far as up.time is concerned, it will continue to treat the tables as if they are regular tables, but we can now manage the data in a much more efficient manner.

MAINTENANCE STEPS

Add Automated Maintenance Procedures

Now that we've converted all the performance tables to partitioned tables and added the necessary triggers for the tables, we can now add the procedures that will automate partition trimming. These database procedures will be triggered on a daily basis by the Event Scheduler built into MySQL 5.1. This means that once we complete the following step, old partitions are automatically removed without any user intervention.

To add the automated maintenance procedures:

- Simply execute the script contents of "4_maint_procs.sql"
- If it ran successfully you should see the following output:

```
...
Query OK, #### rows affected (0.23 sec)
...
Query OK, #### rows affected (0.23 sec)
```

If there are any errors in the output please contact uptime support (support@uptimesoftware.com) with the error(s) provided and we will be glad to help you resolve the issue.

Monitor MySQL Event Scheduler

Since the automated maintenance procedures are executed by the Event Scheduler thread in MySQL, we need actively monitor that thread in case anything should happen to it. To help us do this we can easily setup a quick monitor in uptime (or in a self-monitoring uptime deployment) that monitors the Event Scheduler.

In a production or self-monitoring deployment of up.time, simply create a new "MySQL Basic" service monitor and enter the following settings:

- Name: Automated Table Partition Maintenance Thread
- Port: 3308
- Port Check: (unchecked)
- Username: uptime
- Password: uptime
- Database: uptime
- Script File: (leave empty)
- Script: show variables where variable_name = 'event_scheduler';
- Match: event_scheduler ON

Once the monitor is added and has been tested and is "OK", we can attach an Alert Profile and/or Action Profile so it alerts the appropriate people. You can also have it run a script via an Action Profile to try and start the Event Scheduler if it is off. This is fully configurable by you depending on your internal policies and preferences.

MANAGING THE PARTITIONS

Information

After running all the steps above and verifying that there were no errors all of the performance tables should now be partitioned properly.

The automated maintenance procedures we provide will do a one-month sliding-window transition of older data. It will drop the oldest partition in each performance table and also switch in a new partition for next month. This process is executed on a daily schedule automatically but will only make changes once a month. If there are any questions on this feel free to contact support (support@uptimesoftware.com) for any clarifications.

Running Checks on the Partitions

Below are some helpful queries to verify different things in the table partitions.

View All Partitions in a Table

The following query will list all partitions for a specific table:

```
SELECT table_name, partition_name, partition_ordinal_position,
partition_description, table_rows
FROM information_schema.partitions
WHERE table_name like 'performance_aggregate_par';
```

With this you can see all partitions in a table and how many rows they currently hold.

Check What Partition Will Be Used

The following query will query a partitioned table with a specific date value and return the partition number that will be used when a row with that date will be inserted. It is useful to verify which partition holds or will hold which month of performance data.

```
EXPLAIN PARTITIONS SELECT count(*) FROM performance_sample WHERE sample_time >
'2010-11-01' AND sample_time < '2010-12-01';
```

Just substitute the “function_sample” with the appropriate partition function for the table that you want to check for. For the list of partition functions look at the Technical Information section above.

Dropping Partitions

Dropping a partition from a table will delete that month of data for that table.

```
Alter table <table_name> drop partition par-2011-02;
```

Reorganizing Partitions

The "par_max" partition in every partitioned table should always have zero rows of data; it is a catch-all partition in case the maintenance procedure fails. If there are rows of data in the "par_max" partition we can re-sort them to new partitions with a "REORGANIZE" SQL command.

Example of when a manual reorganize would need to be done:
Current date: July 2011
Number of months to keep in DB: 1
Months of data actually in DB: 5

```
mysql> SELECT table_name, partition_name, partition_ordinal_position, partition_
description, table_rows
    -> FROM information_schema.partitions
    -> WHERE table_name like 'performance_fscap_par'\G
*************************** 1. row ***************************
               table_name: performance_fscap_par
           partition_name: par_2011_02
partition_ordinal_position: 1
     partition_description: 734562
               table_rows: 2242545
*************************** 2. row ***************************
               table_name: performance_fscap_par
           partition_name: par_2011_03
partition_ordinal_position: 2
     partition_description: 734593
               table_rows: 1434609
*************************** 3. row ***************************
               table_name: performance_fscap_par
           partition_name: par_2011_04
partition_ordinal_position: 3
     partition_description: 734623
               table_rows: 1283392
*************************** 4. row ***************************
               table_name: performance_fscap_par
           partition_name: par_max
partition_ordinal_position: 4
     partition_description: MAXVALUE
               table_rows: 2921496
4 rows in set (1.02 sec)
```

- 4+ months of data, and the "par_max" partition has data in it (it should always be empty and never have any rows!)

```
mysql> ALTER TABLE performance_fscap_par REORGANIZE PARTITION par_max INTO (
    ->     PARTITION par_2011_05 VALUES LESS THAN (TO_DAYS('2011-06-01')),
    ->     PARTITION par_2011_06 VALUES LESS THAN (TO_DAYS('2011-07-01')),
    ->     PARTITION par_2011_07 VALUES LESS THAN (TO_DAYS('2011-08-01')),
    ->     PARTITION par_2011_08 VALUES LESS THAN (TO_DAYS('2011-09-01')),
    ->     PARTITION par_2011_09 VALUES LESS THAN (TO_DAYS('2011-10-01')),
    ->     PARTITION par_max VALUES LESS THAN (MAXVALUE)
    -> );
Query OK, 2920823 rows affected (1 min 32.39 sec)
Records: 2920823  Duplicates: 0  Warnings: 0
```

- Run the reorganize command to split the data in "par_max" to their proper partitions.

```
mysql> SELECT table_name, partition_name, partition_ordinal_position, partition_
description, table_rows
    -> FROM information_schema.partitions
    -> WHERE table_name like 'performance_fscap_par'\G
*************************** 1. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_02
partition_ordinal_position: 1
     partition_description: 734562
           table_rows: 2280977
*************************** 2. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_03
partition_ordinal_position: 2
     partition_description: 734593
           table_rows: 1392766
*************************** 3. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_04
partition_ordinal_position: 3
     partition_description: 734623
           table_rows: 1288613
*************************** 4. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_05
partition_ordinal_position: 4
     partition_description: 734654
           table_rows: 1105401
*************************** 5. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_06
partition_ordinal_position: 5
     partition_description: 734684
           table_rows: 966635
*************************** 6. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_07
partition_ordinal_position: 6
     partition_description: 734715
           table_rows: 847007
*************************** 7. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_08
partition_ordinal_position: 7
     partition_description: 734746
           table_rows: 0
*************************** 8. row ***************************
           table_name: performance_fscap_par
       partition_name: par_2011_09
partition_ordinal_position: 8
     partition_description: 734776
           table_rows: 0
*************************** 9. row ***************************
           table_name: performance_fscap_par
       partition_name: par_max
partition_ordinal_position: 9
     partition_description: MAXVALUE
           table_rows: 0
9 rows in set (2.09 sec)
```

- Verified that the partitions are created and that there is no data in "par_max"