

UP.TIME DATA MANAGEMENT PACK:

SQL SERVER 2005 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: SQL SERVER TABLE PARTITIONING	4
SELECTING THE APPROPRIATE PARTITION TYPE.....	4
TECHNICAL INFORMATION	5
REQUIREMENTS	5
LIMITATIONS	5
TABLE PARTITION TYPES USED.....	5
PRE-SETUP INSTRUCTIONS	7
SQL SERVER USER PERMISSIONS	7
STOP ALL UP.TIME SERVICES	7
CONVERSION INFO	8
BACKUP THE DATABASE	8
PRE-REQUISITE(S).....	8
CONVERSION STEPS.....	8
HOW TO EXECUTE THE SCRIPTS.....	8
CONVERSION STEPS	10
1. CREATE PARTITION FUNCTIONS AND SCHEMES.....	10
2. CREATE NEW PARTITIONED TABLES	10
3. COPY DATA TO NEW TABLES	10
4. CREATE TRIGGERS	11
5. CREATE INDEXES	11
VERIFYING THE TABLES ARE PARTITIONED PROPERLY.....	12
DISABLING ARCHIVING IN UP.TIME	12
MANAGING THE PARTITIONS	13
INFORMATION.....	13
TECHNICAL PARTITION INFORMATION	13

MANAGING THE PARTITIONS IN A SLIDING WINDOW SCENARIO 13

STEPS FOR SWITCHING OUT AN OLD PARTITION (REMOVING OLD DATA)..... 14

STEPS FOR SWITCHING IN A NEW PARTITION (FOR NEW DATA) 14

REVISION HISTORY

Rev	Date	Author	Description
1	Sep. 15, 2009	Joel Pereira	Initial Publication
2	Sep. 22, 2009	Joel Pereira	Clarifications in "managing partitions" section
3	Sep. 23, 2009	Joel Pereira	Updated support link

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
- SQL Server Transaction Log filled (during archiving)

The Solution: SQL Server 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 SQL Server 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 SQL Server 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://msdn.microsoft.com/en-us/library/ms345146\(SQL.90\).aspx](http://msdn.microsoft.com/en-us/library/ms345146(SQL.90).aspx)

TECHNICAL INFORMATION

Requirements

- SQL Server 2005 Enterprise

Limitations

- Only works on SQL Server 2005 Enterprise (and up)
- **Partitions need to be manually added to all the tables below on a monthly/yearly scheduled basis, and the last partition (the last partition with the date set furthest into the future) should always be empty. If a row is inserted into the last partition, the next time when we switch in a new table, it may take a long time for the command to execute because it must now parse all the data in the last partition.**
For more info on this please read the Microsoft knowledgebase link above regarding table partitioning.

Table Partition Types Used

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

Table Name	Type of Partitioning
performance_sample	Range Partitioning
performance_aggregate	Range Partitioning
performance_cpu	Range Partitioning
performance_disk	Range Partitioning
performance_esx3_workload	Range Partitioning
performance_fscap	Range Partitioning
performance_lpar_workload	Range Partitioning
performance_network	Range Partitioning
performance_nrm	Range Partitioning
performance_psinfo	Range Partitioning
performance_vxvol	Range Partitioning
performance_who	Range Partitioning
erdc_int_data	Range Partitioning
erdc_decimal_data	Range Partitioning
erdc_string_data	Range Partitioning
ranged_object_value	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. 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.

PRE-SETUP INSTRUCTIONS

SQL Server User Permissions

The user that up.time uses to connect to SQL Server (in the <uptime_dir>/uptime.conf file) must be able to manage the tables in the up.time schema as well as setup table partitioning.

Stop All 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"

How to stop up.time on Linux/Solaris:

- `/etc/init.d/uptime_core stop`

CONVERSION INFO

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.

Pre-Requisite(s)

The most important thing to do before running any of the scripts is to make sure that you 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 2 tables in your database it should be more than 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.

Conversion Steps

Below are the steps that we will take to convert the tables that hold all performance data in the up.time schema into partitioned tables.




Note: Remember to backup your entire database before proceeding any further!

1. Create partition functions and schemes
2. Create new partitioned tables
3. Copy data from old tables to new ones
4. Create triggers
5. Create indexes

How to Execute the Scripts

The scripts that will perform the conversion to partitioned tables are meant to be executed via the SQL Server Management Studio interface.

To run a script:

- Use SQL Server Management Studio to connect to your SQL Server database and click on the "New Query" button in the menu bar ()
- Open the specified script in Notepad and copy & paste the entire contents of the script into the query section of Management Studio
- The scripts assume that the uptime database is set to "uptime"; if this is not the case, edit the first line that states "USE uptime" to instead use your database name where the uptime data is stored.
- Click on the Parse button () to verify if the script syntax is correct, and then click on the Execute button () to execute the query

CONVERSION STEPS

1. Create Partition Functions and Schemes

The partition function and schemes define which column we will use for partitioning as well as the value range we will be using (monthly) and where each partition will be stored (if on different filegroups/disks). The scripts included will create monthly partitions and will store all the data in the "PRIMARY" filegroup.

Since the partition functions must include up-to-date date ranges, we have an online resource setup to provide current output that can be used to create updated partition functions/schemes.

- Access the following link to generate updated partition functions/schemes:
Link: <http://support.uptimesoftware.com/tools/partitioning-sql-generator.php?partitions=12>
- Edit the "partitions=" section of the link to have the page generate more/less partitions, depending on how many months of data you want to store in the database
- Copy & paste the generated SQL commands into the Management Studio query section and execute the commands
- If it ran successfully you should see the following output:

```
Command(s) completed successfully.
```

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

2. 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 date/datetime field.

To create the new partitioned tables:

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

```
Command(s) completed successfully.
```

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

3. Copy Data to New 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 all of these steps one table at a time.

Example:

- It will copy all the data from the "performance_sample" table into the "performance_sample_p" table
- Once that is done, it will drop the "performance_sample" table
- Then it will rename the "performance_sample_p" table to just "performance_sample"

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 aware of the changes.

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

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

```
(##### row(s) affected)
Caution: Changing any part of an object name could break scripts and stored
procedures.

(##### row(s) affected)
Caution: Changing any part of an object name could break scripts and stored
procedures.

(##### row(s) affected)
Caution: Changing any part of an object name could break scripts and stored
procedures.

...
```

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

4. Create Triggers

Due to the way partitioned tables work in SQL Server, all tables must have a date/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 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 that problem we will use triggers to have the database take care of setting that column.

To add the necessary triggers to the database:

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

```
Command(s) completed successfully.
```

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

5. Create Indexes

Now that we've converted all the performance tables to partitioned tables and added the necessary triggers for the tables, we can now create the indexes so that up.time can access the data as quickly as possible.

To create the indexes:

- Simply execute the script contents of "5. create_indexes.sql"
- If it ran successfully you should see the following output:

```
Command(s) completed successfully.
```

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

Verifying the Tables are Partitioned Properly

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

If you have data in your database you can run the following SQL query to have a look at how many rows of data are in each partition in the performance_sample table:

```
SELECT $PARTITION.function_sample(sample_time) AS PartitionNum, COUNT(*) AS [COUNT]
FROM performance_sample
GROUP BY $PARTITION.function_sample(sample_time)
ORDER BY PartitionNum
```

To run the same query for the other tables just change the text in red for the other tables.

- function_sample (partition function for the performance_sample table)
- sample_time (partitioning column used for the performance_sample table)
- performance_sample (the performance table we're querying)

Check below for a chart with the above info for each table that we partitioned.

The following query will return the partition number that will be used when requesting/inserting data for the date range specified. You can use this query to verify which partition number contains which month of performance data.

```
SELECT $PARTITION.function_sample('2009-10-01')
```

Disabling Archiving in up.time

Now that we've converted all the performance tables to partitioned tables we can disable the built-in archiving procedure in up.time so that it no longer runs. Archiving older data will now be managed by the DBA(s) in your company and not by the up.time application. If archiving is not disabled it will try to run on the first of every month and possibly cause performance issues if you have very large performance tables.

How to disable archiving:

- Login to the up.time interface
- Click on the Config tab, and then click on the Archive Policy link on the left menu
- Make sure the "Enable Archiving" checkbox is unchecked
- Click on the "Set Archive Policy" button

That's it; archiving should now be disabled in up.time!

MANAGING THE PARTITIONS

Information

Before proceeding further we highly recommend reading the following Microsoft knowledgebase article describing how SQL Server table partitioning works and the various ways of managing the partitions. The database will have to be managed on a monthly/yearly schedule (depending on your own policy that you setup) by your DBA's and they must be aware of the steps that need to be taken as well as any issues that could arise if the appropriate steps are not taken.

Resource : [http://msdn.microsoft.com/en-us/library/ms345146\(SQL.90\).aspx](http://msdn.microsoft.com/en-us/library/ms345146(SQL.90).aspx) (msdn.microsoft.com)

Note: We cannot provide simple-to-run scripts that will always work with everyone's different configuration as it varies on a lot of different factors. We have only provided the tools to help your DBAs create your own maintenance scripts. If there are any questions on this feel free to contact support (support@uptimesoftware.com) for any clarifications.

Technical Partition Information

Each partition will contain 1 month of data for the table it is a part of.

Table Partition Information in the up.time Schema:

Table Name	Partitioning Column	Partition Function	Partition Scheme
performance_sample	sample_time	function_sample	scheme_sample
performance_aggregate	sample_date	function_aggregate	scheme_aggregate
performance_cpu	sample_date	function_cpu	scheme_cpu
performance_disk	sample_date	function_disk	scheme_disk
performance_esx3_workload	sample_date	function_esx3_workload	scheme_esx3_workload
performance_fscap	sample_date	function_fscap	scheme_fscap
performance_lpar_workload	sample_date	function_lpar_workload	scheme_lpar_workload
performance_network	sample_date	function_network	scheme_network
performance_nrm	sample_date	function_nrm	scheme_nrm
performance_psinfo	sample_date	function_psinfo	scheme_psinfo
performance_vxvol	sample_date	function_vxvol	scheme_vxvol
performance_who	sample_date	function_who	scheme_who
erdc_int_data	sampletime	function_int_data	scheme_int_data
erdc_decimal_data	sampletime	function_decimal_data	scheme_decimal_data
erdc_string_data	sampletime	function_string_data	scheme_string_data
ranged_object_value	sample_time	function_ranged_object_value	scheme_ranged_object_value

Managing the Partitions in a Sliding Window Scenario

The steps below are simply recommendations for your DBA(s) on managing the data in a "sliding window" scenario, which is described in the link above. It is the responsibility of your DBA(s) to manage the data on a scheduled basis.

"Managing the data" includes the following:

- "Switch Out" the oldest partition (table) for deletion/archival

- “Switch In” a new partition (table) for new data

Steps for Switching Out an Old Partition (removing old data)

The steps to archive/remove the oldest month out of a partitioned table in the up.time schema involve the following:

- Create a new table that is identical to the current one (minus any partitioning)
- Get the oldest partition number
- Alter the current table to “switch out” the partition to the new empty table
- Alter the partition function to merge the last partition (now empty) with the second last partition (next month)

“Create Table” Statements

As mentioned above, you will need to create a new table to switch the last partition into. The new table will contain the oldest month of data in the database. You can then choose to archive the data somewhere or simply delete it from the database; this is completely up to you and/or your company policy.

The CREATE TABLE SQL statements that you can use for switching out the oldest month are included in the sql file named “maint-create_tables.sql”. You can copy the SQL for each table from that file, or simply run the entire script as-is without a problem and it will create the empty tables for you which you can then use for switching out the oldest partition. It will create the new tables with names appended by “_old”, such as “performance_sample_old”.

Note: The script only contains “CREATE TABLE” commands and will not perform anything else. To manage the partitioned tables you will need to use the example(s) below and have your DBAs create maintenance scripts as per your own internal data policy.

SQL Commands to Switch Out the Oldest Month

Example SQL to switch out the oldest partition in the “performance_sample” table to the empty table “performance_sample_old”:

```
BEGIN TRANSACTION

DECLARE @OLDEST_PARTITION_NUM numeric(5)
SELECT @OLDEST_PARTITION_NUM = min($PARTITION.function_sample(sample_time)) FROM
performance_sample

ALTER TABLE performance_sample SWITCH PARTITION @OLDEST_PARTITION_NUM TO
[performance_sample_old]PARTITION @OLDEST_PARTITION_NUM

ALTER PARTITION FUNCTION [function_sample] () MERGE RANGE (N'2009-04-01')

COMMIT TRANSACTION
```

Red text indicates the values that must be changed/updated.

Steps for Switching In a New Partition (for new data)

As per the Microsoft KB article, the last partition (date is furthest in the future) should always be empty so that this operation does not handle any actual row data and is strictly a meta-data operation. This will ensure that this operation is very fast and does not impact database performance in any negative way.

Switching In a new partition involves the following steps:

1. Alter the partition scheme and add a new filegroup for the new partition

2. Alter the partition function and add a new partition by splitting the last partition (meta-data operation)
3. Repeat steps 1-2 for every partitioned table in the list above

Example SQL to switch in a new partition for Nov. 2009 data on the “performance_sample” table:

```
BEGIN TRANSACTION

ALTER PARTITION SCHEME [scheme_sample] NEXT USED [PRIMARY]
ALTER PARTITION FUNCTION [function_sample] () SPLIT RANGE (N'2009-12-01')

COMMIT TRANSACTION
```

Red text indicates the values that must be changed/updated.

Note: Remember that you will need to execute the same SQL commands above for each partitioned table.