PANDUIT™ infrastructure for a connected world

smartzone™

TECHNICAL REFERENCE 91-SZ

Published: **7/25/18**

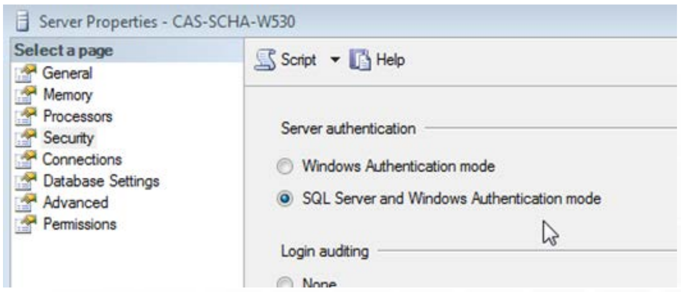
# SmartZone Database Setup and Maintenance

| | |
|---|---|
| **Applies to:** | SmartZone Database |
| **Objective:** | This Technical Reference serves as a checklist of Database Administrator tasks for setting up and maintaining a SmartZone database. |
| **Pre-Requisites:** | Optimize SQL Server by changing settings. |

## Performing the Procedure

Before you create the SmartZone database via SmartZone Install, optimize SQL Server by changing these settings.

### Mixed Mode



Server authentication
- Windows Authentication mode
- [✓ SQL Server and Windows Authentication mode]

The Database instance should be set to Mixed Mode where the SmartZone Database (acmnms) resides.

### Increasing Memory

The default SQL Server settings do not aggressively allocate memory. This situation significantly impacts performance on most deployments of a SQL Server database.

To increase the memory for SQL Server:

1. Using Microsoft SQL Server Management Studio, connect to the database server where you want to host the SmartZone database.
2. **Right-click** the database connection and select Properties.
3. Select the Memory page and enter a size in (in MB) the Minimum Server Memory box that is equal to the size of the free memory you have on the server.

## Setting Processor Priority

On dedicated database servers (recommended for production installations of SmartZone), the SQL Server process is configured to prevent over-consumption of the system CPU resources.

To set the processor priority:

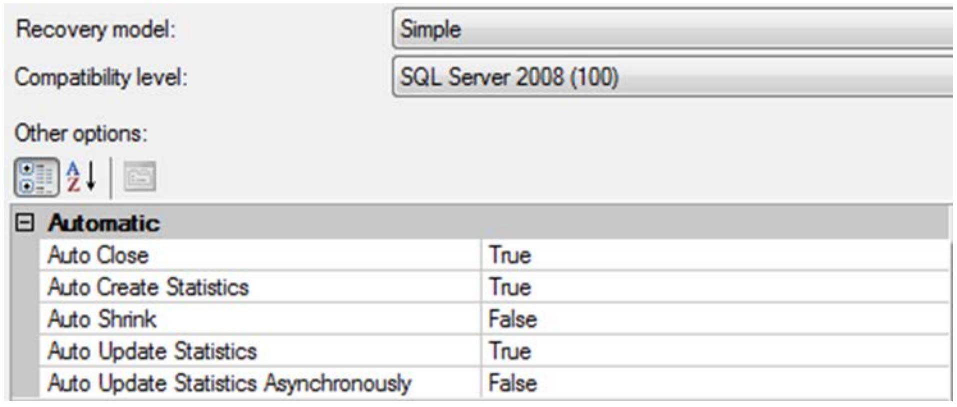
1. Use Microsoft SQL Server Management Studio to connect to the database server where you want to host the SmartZone database.
2. **Right-click** the database connection and select **Properties**.
3. **Select** the Processors page and select **Boost SQL Server Priority**.

After the SmartZone installs, optimize the SQL Server by changing these settings.

## Increasing the Recovery Interval

This setting specifies the amount of time the deployment waits for recovery after a crash. The SQL Server default setting is 1 minute. Increasing this setting to a larger value improves performance because it allows the server to be more relaxed in writing changes from the database log to the database files.

*Recovery Mode (simple) and Auto Shrink (false)*



| | |
|---|---|
| Recovery model: | Simple |
| Compatibility level: | SQL Server 2008 (100) |

Other options:

| Automatic | |
|---|---|
| Auto Close | True |
| Auto Create Statistics | True |
| Auto Shrink | False |
| Auto Update Statistics | True |
| Auto Update Statistics Asynchronously | False |

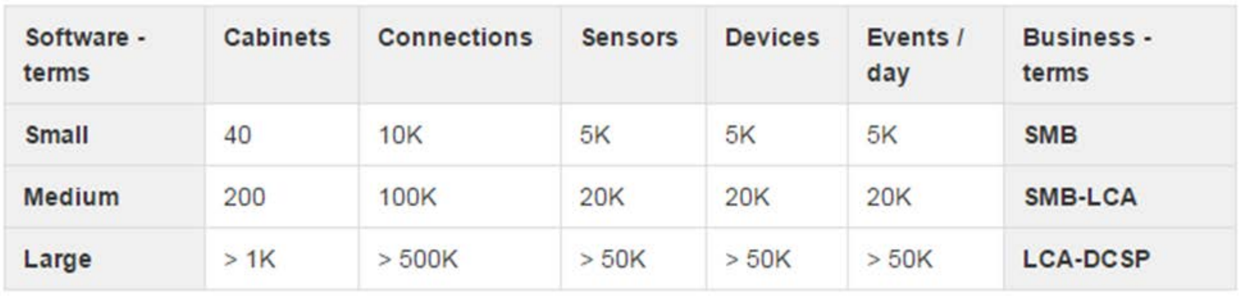
To increase the recovery interval:

1. Use Microsoft SQL Server Management Studio to connect to the database server where you want to host the SmartZone database.
2. **Right-click** the database connection and **select** Properties.
3. **Select** the Database Settings page and **type** 5 in the Recovery Interval (Minutes) box.

**NOTE**: After you change these settings, restart the SQL Server database to ensure that the settings take effect.

## Sizing Your SQL Server Database

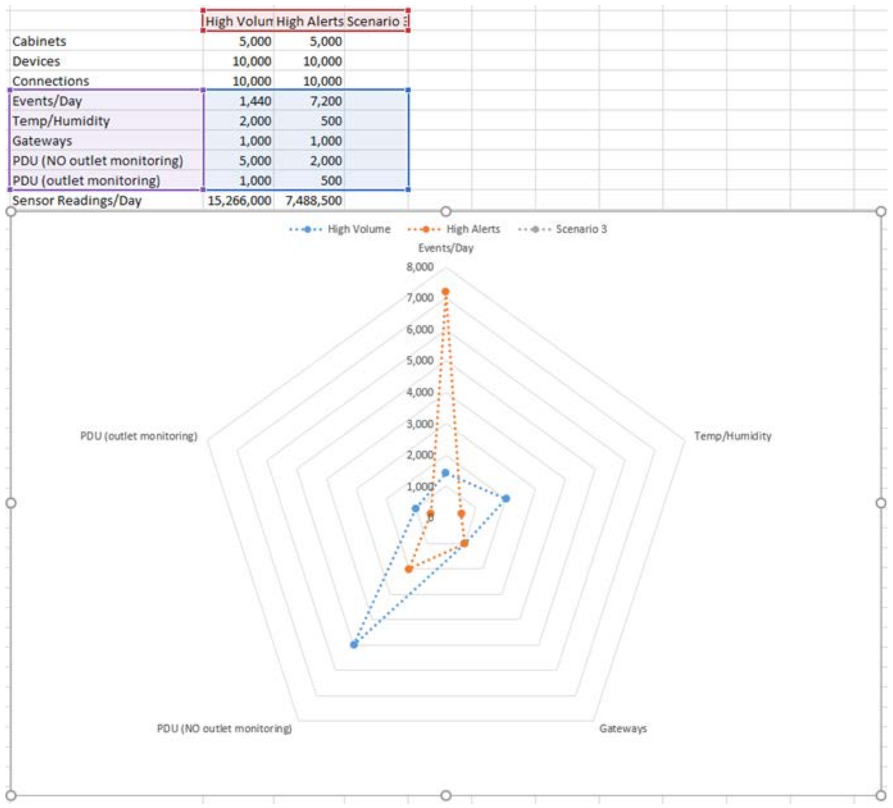
The default database sizes that SQL Server provides are too small for SmartZone. It is best to pre-allocate the database size at creation to reflect your deployment requirements:



| Software - terms | Cabinets | Connections | Sensors | Devices | Events / day | Business - terms |
|---|---|---|---|---|---|---|
| **Small** | 40 | 10K | 5K | 5K | 5K | **SMB** |
| **Medium** | 200 | 100K | 20K | 20K | 20K | **SMB-LCA** |
| **Large** | > 1K | > 500K | > 50K | > 50K | > 50K | **LCA-DCSP** |

- **Small size deployments**: Set Database Data Initial Size to 1 GB, and set auto-grow to 250 MB.
- **Small-Medium size deployments**: Set Database Data Initial Size to 2 GB, and set auto-grow to 1 GB.
- **Medium-Large size deployments**: Not recommended for SQL Express.

Database growth is always restricted to a certain size. Monitor the resource usage of the SmartZone database to ensure that it does not run out of restricted space or the space available on the disks where the database resides.



| | High Volun | High Alerts | Scenario : |
|---|---|---|---|
| Cabinets | 5,000 | 5,000 | |
| Devices | 10,000 | 10,000 | |
| Connections | 10,000 | 10,000 | |
| Events/Day | 1,440 | 7,200 | |
| Temp/Humidity | 2,000 | 500 | |
| Gateways | 1,000 | 1,000 | |
| PDU (NO outlet monitoring) | 5,000 | 2,000 | |
| PDU (outlet monitoring) | 1,000 | 500 | |
| Sensor Readings/Day | 15,266,000 | 7,488,500 | |

Boundaries of typical SmartZone installs on SQL Express need some considerations:

- How many millions of sensor records per day?
- How many power or temperature threshold alerts per minute?
- How many PDUs is the data polled from?
- How many Gateways is the data polled from?
- Are the PDUs enabled for per outlet monitoring?
- Is a Data Retention plan in effect and for how many days of data?

PANDUIT™

## Excessive Indexing

```
-- Possible Bad NC Indexes (writes &gt; reads)

 SELECT OBJECT_NAME(s.[object_id]) AS [Table Name], i.name AS [Index Name],
i.index_id,

 user_updates AS [Total Writes], user_seeks + user_scans + user_lookups AS [Total
Reads],

 user_updates - (user_seeks + user_scans + user_lookups) AS [Difference]

 FROM sys.dm_db_index_usage_stats AS s WITH (NOLOCK)

 INNER JOIN sys.indexes AS i WITH (NOLOCK)

 ON s.[object_id] = i.[object_id]

 AND i.index_id = s.index_id

 WHERE OBJECTPROPERTY(s.[object_id],'IsUserTable') = 1

 AND s.database_id = DB_ID()

 AND user_updates > (user_seeks + user_scans + user_lookups)

 AND i.index_id > 1

 ORDER BY [Difference] DESC, [Total Writes] DESC, [Total Reads] ASC;
```

## Plan Cache Bloat

You should not see an excessive number of query plans in the plan cache that are only used once (i.e., ad-hoc plans) because these plans take up memory that could otherwise be used to store data pages.

```
WITH CACHE_ALLOC AS

(

SELECT objtype AS [CacheType]

       ,COUNT_BIG(objtype) AS [Total Plans]

       ,sum(cast(size_in_bytes as decimal(18,2)))/1024/1024

       AS [Total MBs]

       , avg(usecounts) AS [Avg Use Count]

       , sum(cast(

           (CASE WHEN usecounts = 1

           THEN size_in_bytes

           ELSE 0

           END)

          AS decimal(18,2)))/1024/1024
```

```
        AS [Total MBs - USE Count 1]
        ,   CASE
            WHEN (Grouping(objtype)=1) THEN count_big(objtype)
            ELSE 0
            END AS GTOTAL
        FROM sys.dm_exec_cached_plans
GROUP BY objtype
)
   SELECT
       [CacheType], [Total Plans],[Total MBs],
       [Avg Use Count],[Total MBs - USE Count 1],
       Cast([Total Plans]*1.0/Sum([Total Plans])OVER() * 100.0 AS DECIMAL(5, 2))
       AS Cache_Alloc_Pct
       FROM CACHE_ALLOC
       Order by [Total Plans] desc
```

You can identify problems by checking for an excessive number of compilations. There should not be many of these ad-hoc plans getting stored in the plan cache without them all being compiled. Here's some quick measurements to see if the system is having an excessive number of recompiles. The defined acceptable level is 10% of the batch requests being compilations.

```
select t1.cntr_value As [Batch Requests/sec],
    t2.cntr_value As [SQL Compilations/sec],
    plan_reuse =
    convert(decimal(15,2),
    (t1.cntr_value*1.0-t2.cntr_value*1.0)/t1.cntr_value*100)
from
    master.sys.dm_os_performance_counters t1,
    master.sys.dm_os_performance_counters t2
where
    t1.counter_name='Batch Requests/sec' and
    t2.counter_name='SQL Compilations/sec
```

There is a simple way to get rid of the cached execution plans and other related information by executing:

- DBCC FREEPROCCACHE
- DBCC DROPCLEANBUFFERS

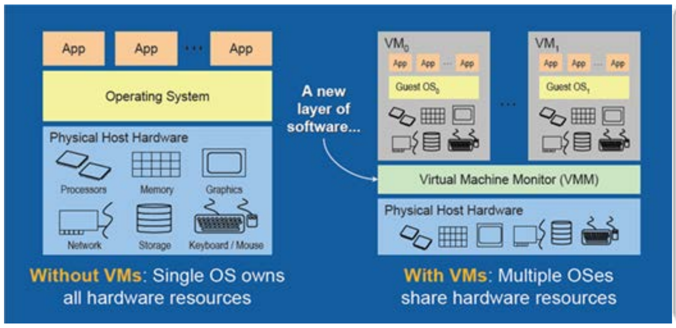
## Transaction Log and Data

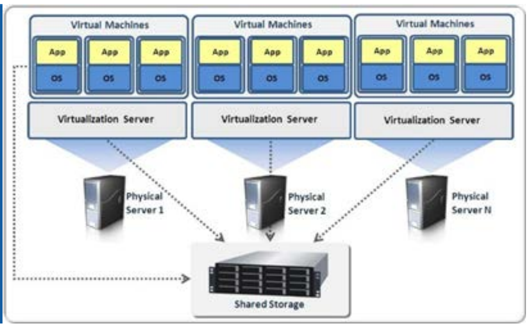
It is preferable to have separate dedicated drives for data and log files. Not only is it good for performance, but it helps to mitigate disasters by keeping the transaction logs on a different drive from the data.

- DBCC SQLPERF(logspace)
- DBCC LOGINFO
- DBCC OPENTRAN
- DBCC INPUTBUFFER(<SPID Value>)
- DBCC SHRINKFILE(acmnms_log,200)

## Virtualization

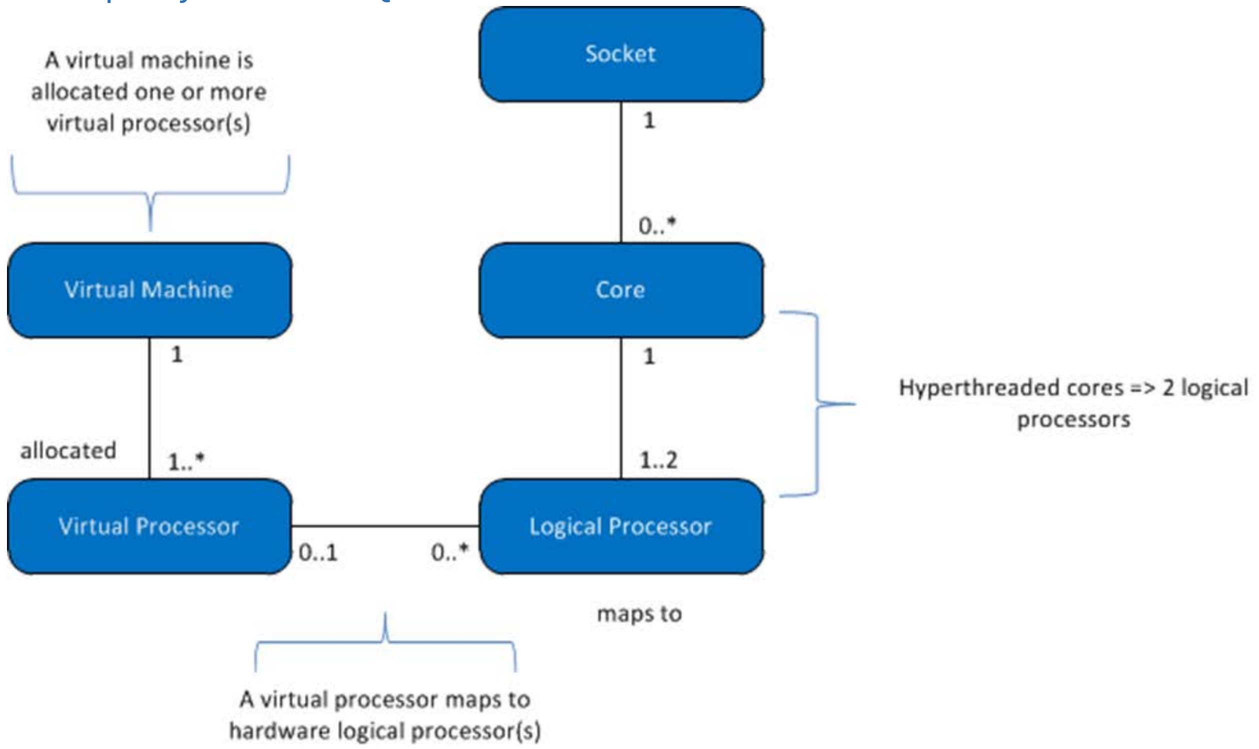
*Simple (compute & storage all in one)* *Complex (separate compute & storage)*


App
App
App
Operating System
Physical Host Hardware
Processors
Memory
Graphics
Network
Storage
Keyboard / Mouse
A new layer of software...
VM0
Guest OS0
VM1
Guest OS1
Virtual Machine Monitor (VMM)
Physical Host Hardware
Without VMs: Single OS owns all hardware resources
With VMs: Multiple OSes share hardware resources

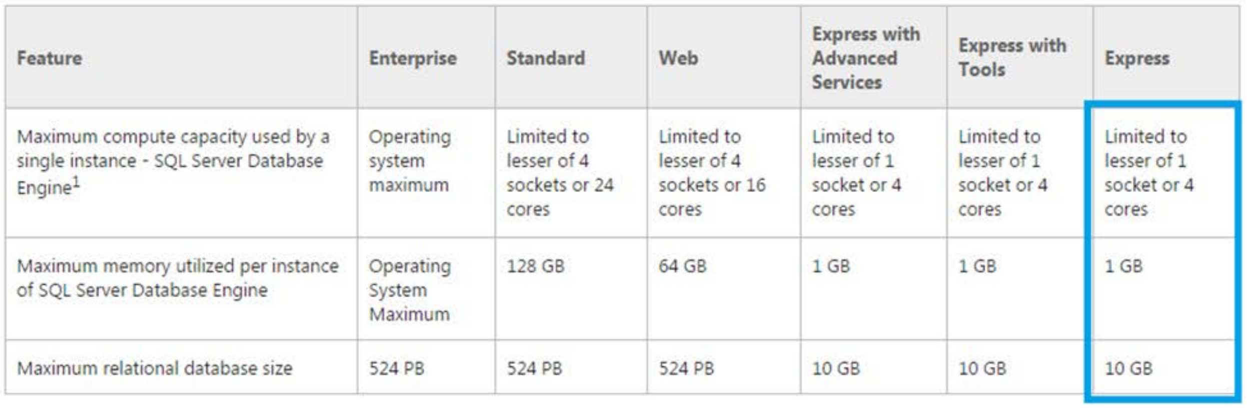


*Use Sql Server Activity Monitor.run stored procedures directly in Enterprise Manager to monitor execution time.*

## Compute Capacity Limits of SQL Server


A virtual machine is allocated one or more virtual processor(s)
Socket
1
0..*
Virtual Machine
Core
1
1
Hyperthreaded cores => 2 logical processors
allocated
1..*
1..2
Virtual Processor
Logical Processor
0..1
0..*
maps to
A virtual processor maps to hardware logical processor(s)

https://msdn.microsoft.com/en-us/library/ms143760.aspx



| Feature | Enterprise | Standard | Web | Express with Advanced Services | Express with Tools | Express |
|---|---|---|---|---|---|---|
| Maximum compute capacity used by a single instance - SQL Server Database Engine[1] | Operating system maximum | Limited to lesser of 4 sockets or 24 cores | Limited to lesser of 4 sockets or 16 cores | Limited to lesser of 1 socket or 4 cores | Limited to lesser of 1 socket or 4 cores | Limited to lesser of 1 socket or 4 cores |
| Maximum memory utilized per instance of SQL Server Database Engine | Operating System Maximum | 128 GB | 64 GB | 1 GB | 1 GB | 1 GB |
| Maximum relational database size | 524 PB | 524 PB | 524 PB | 10 GB | 10 GB | 10 GB |

https://msdn.microsoft.com/en-us/library/cc645993.aspx

## Database Maintenance

The following sections lay out the Database Maintenance Plan.

## Configuring the Recovery Model

The Transaction Log ensures that only valid data is written out to the database via rollback, and it allows transactions to be played back to recreate the system state right before a failure.

Each SQL Server database includes a Recovery Model, which determines:

- How transactions are logged
- Whether the transaction log can be backed up
- The type of restore operations permitted

By default, a new database inherits the recovery model from the model database. However, you can override the default setting by assigning one of the following three recovery models.

- **Simple**: In this model, transaction log backups are not permitted. The model automatically reclaims log space, so there is almost no need to manage the transaction log space. However, this is also the riskiest of the models – a database can be restored only to the point of its last backup. This model is generally used for the system databases, and for both testing and development, although it is sometimes appropriate for a read-only database such as a data warehouse.
- **Full**: The log files can and should be backed up, as they provide full recovery to a specific time. This model is less risky than the Simple model. In the Full recovery model, all operations are fully logged, including bulk import operations. The Full recovery model is generally the model used for production environments.
- **Bulk Logged**: This model is intended as an adjunct to the Full recovery model, because operations such as bulk import are minimally logged. You do not need to log these transactions, because you can reload the data if necessary. In such cases, users can set the recovery model to Bulk Logged while importing the data, and then change the setting back to Full when user is finished. (**Note**: You should perform a full back up after changing the setting back to Full.)

You can switch the recovery model on a database by running an ALTER DATABASE statement and specifying the SET RECOVERY clause, as shown in the following example:

```
USE master;

ALTER DATABASE acmnms SET RECOVERY FULL;
```

By default, the model database is configured with the Full recovery model, which means that the acmnms was automatically configured with the Full model because it inherited the setting from the model database.

## Monitoring the Log File

When maintaining a database's transaction log, you may want to retrieve information about the log so you can verify its settings or track how much log space is being used. One way to find information about the log is by using the sys.database_files catalog view. The view returns details about database files, including:

- File type
- Current file size
- Maximum file growth

In the following example, the sys.database_files catalog view is used to retrieve data about the log file associated with the acmnms database:

```
USE acmnms;

SELECT name,
  size, -- in 8-KB pages
  max_size, -- in 8-KB pages
  growth,
  is_percent_growth
FROM sys.database_files
WHERE type_desc = 'LOG'
```

The statement returns:

- Current file size
- Maximum file growth
- Growth rate

- The is_percent_growth flag, which determines how the growth rate should be interpreted. If the flag is set to 0, the growth rate is the number of 8-KB pages. If the flag is set to 1, the growth rate is a percentage.

You also can use the DBCC SQLPERF statement to return information about the transaction logs for each database in a SQL Server instance. To retrieve log data, specify the LOGSPACE keyword in parentheses, as shown in the following example:

```
DBCC SQLPERF(LOGSPACE);
```

You also can generate a report in SQL Server Management Studio that graphically displays data like the results of the DBCC SQLPERF statement.

To generate the report

1. **Right-click** the name of the database in Object Explorer.
2. Select Reports.
3. Select Standard Reports.
4. **Click** Disk Usage.

## Backing Up the Log File

If a database is configured with the Full or Bulk Logged recovery model, you should back up the transaction log regularly so it can be truncated to free-up inactive log space. The backup can also be used (along with the database backups) to restore the database in the event of failure.

Before backing up a log file, perform a full database backup. For example, you can run the following BACKUP DATABASE statement on the acmnms database:

```
BACKUP DATABASE acmnms TO DISK = 'E:\DbBackup\acnmnms_dat.bak';
```

Make sure the TO DISK location exists, or specify a different location.

After the database has been backed up, you can back up the transaction log by using the BACKUP LOG statement and specifying the target destination for the backup files, as shown in the following example:

```
BACKUP LOG EmployeeDB TO DISK = 'E:\LogBackup\EmployeeDB_log.bak';
```

After you back up the transaction log, the SQL Server database engine automatically truncates inactive log space. Truncating a log file removes inactive virtual log files but does not reduce the file size. In addition, you cannot specifically truncate a log. You can, however, shrink the file, which does reduce the size. (See Shrinking a Log File on page 13.)

## Modifying a Log File

You can use the ALTER DATABASE statement to modify a log file. Specify the MODIFY FILE clause, along with the appropriate options. In addition to specifying the logical name of the log file, you can define the following three arguments:

**SIZE**: Specify the new size of the log file. The new size must be greater than the current size, or you will receive an error when you run the statement.

**MAXSIZE**: Specify the maximum size that the file can grow to. If you do not specify a maximum size, the file will grow until it fills the disk (assuming the space is needed).

**FILEGROWTH**: Specify the growth increment used when expanding the file. You can specify the size as KB, MB, GB, or TB, or as a percentage, such as 10%. If a number is specified without a suffix, MB is used. If no number is specified, 10% is used. A value of 0 indicates that no automatic growth is allowed.
The following ALTER DATABASE statement modifies the acmnms_log file in the acmnms DB database:

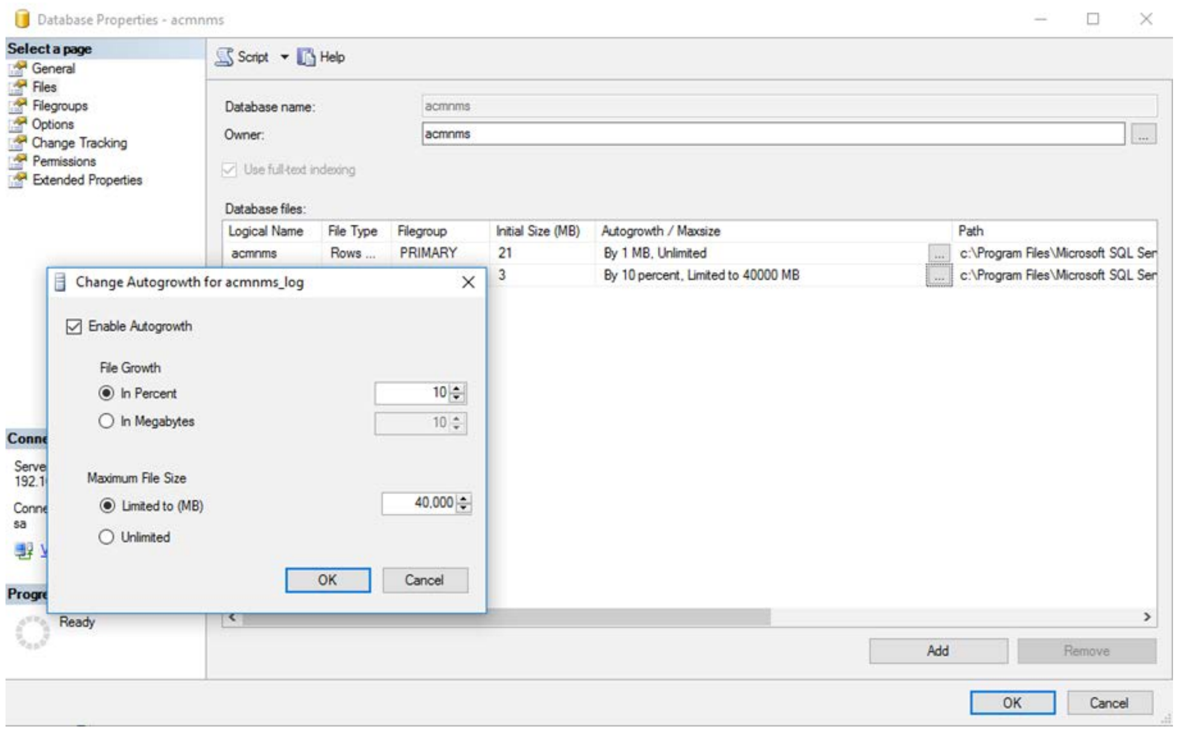
```
ALTER DATABASE acmnms
MODIFY FILE
(NAME = acmnms_log,
    SIZE = 20GB,
    MAXSIZE = 40GB,
        FILEGROWTH = 10%);
```

In this example, the user has specified the logical name of the log file, the new size for the file (2 MB), the maximum size (200 MB), and the growth increment (10 MB).

After running the ALTER DATABASE statement, you can query the sys.database_files catalog view to verify the changes. Results should be like the following:

```
Name        size max_size  growth    is_percent_growth
acmnms_log      256  25600           1280      0
```

The file size is now 256 8-KB pages, the maximum size is 25,600 8-KB pages, and the growth increment is 1,280 8-KB pages.



Database Properties - acmnms

Database name: acmnms

Owner: acmnms

Use full-text indexing

Database files:

| Logical Name | File Type | Filegroup | Initial Size (MB) | Autogrowth / Maxsize | Path |
|---|---|---|---|---|---|
| acmnms | Rows ... | PRIMARY | 21 | By 1 MB, Unlimited | c:\Program Files\Microsoft SQL Ser |
| | | | 3 | By 10 percent, Limited to 40000 MB | c:\Program Files\Microsoft SQL Ser |

Change Autogrowth for acmnms_log

Enable Autogrowth

File Growth

In Percent: 10

In Megabytes: 10

Maximum File Size

Limited to (MB): 40,000

Unlimited

## Shrinking a Log File

To truncate the transaction log, you must first back up the log. The database engine then automatically truncates the inactive records. However, truncating the log doesn't reduce its size. Instead, you must shrink the log file, which removes one or more inactive virtual log files.

To shrink a log file, you can run a DBCC SHRINKFILE statement that specifies the name of the log file and the target size, in MB. For example, the following DBCC SHRINKFILE statement shrinks the acmnms_log file:

```
DBCC SHRINKFILE (acmnms_log, 2000);
```

The target size in this statement is 1 MB (128 8-KB pages). When a user runs the statement, the database engine will shrink the file down to that size, but only if there are enough inactive virtual log files to allow the size reduction.

## End State

You have created a SmartZone database via SmartZone install.