MSSQLTips.com
brought to you by Edgewood Solutions

Your daily source for SQL Server Tips

Best Practices for SharePoint Search Databases

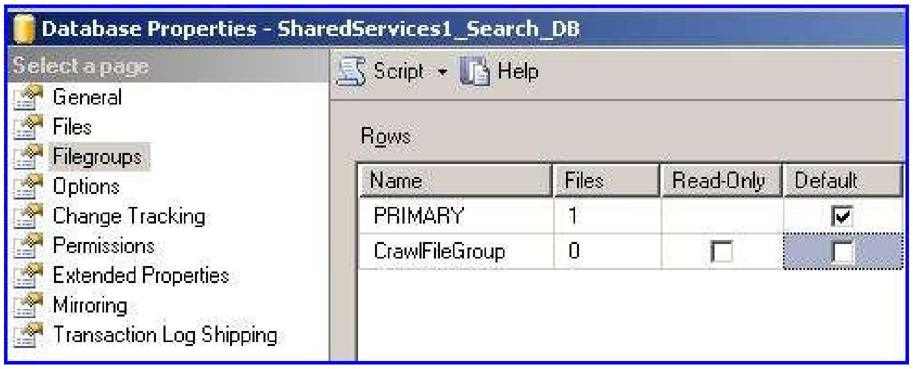
Written By: Ray Barley -- 7/15/2009

Problem
I'm an experienced DBA, but new to administering SharePoint 2007 databases. Our current SharePoint databases are very large and getting bigger every day. I notice the search database is growing fast and also has a lot of activity. Can you give me the details I need to know to implement best practices for this database?

Solution
The search database supports crawling content sources and end-user queries. There are tables and indexes in the database that are used just for crawling; other tables and indexes are used for end-user queries (although the crawling process does updates these). The crawling process itself is very IO intensive. If you are experiencing a heavy IO load on the search database, you can separate the crawling tables from the query tables by putting them in separate filegroups. The use of multiple filegroups is a recognized best practice for a SQL Server database. The idea is simply to spread IO over multiple disks. For the search database, there is a blog post on the Microsoft Enterprise Search Blog that suggests the way to do this is to move the crawling tables and indexes to a new filegroup. In this tip we will review the steps to create a new filegroup and move selected tables and indexes to that filegroup.

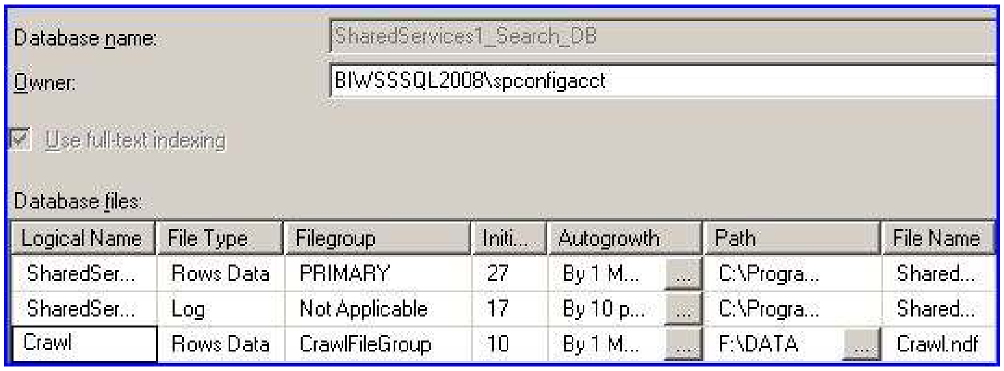
Add a Filegroup to the Search Database

Use SQL Server Management Studio and locate the search database in the Object Explorer. I have a default SharePoint installation, so the name of my search database is SharedServices1_Search_DB; yours may be different. Right click on the search database to display the Database Properties dialog, select Filegroups, then add a filegroup named CrawlFileGroup as shown below:



Name	Files	Read-Only	Default
PRIMARY	1		☑
CrawlFileGroup	0	☐	☐

After creating a new filegroup, you need to add one or more files to it. Select Files from the Database Properties for the search database and add a file as shown below:



Database name: SharedServices1_Search_DB

Owner: BIWSSSQL2008\spconfigacct

☑ Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initi...	Autogrowth	Path	File Name
SharedSer...	Rows Data	PRIMARY	27	By 1 M...	C:\Progra...	Shared...
SharedSer...	Log	Not Applicable	17	By 10 p...	C:\Progra...	Shared...
Crawl	Rows Data	CrawlFileGroup	10	By 1 M...	F:\DATA	Crawl.ndf

Note that the third line in the above screen shot (Logical Name = Crawl) is the file that I added. The Filegroup is CrawlFileGroup, the one that we just added in the previous step. Also note that the path is "D:\DATA" which is different than the Path of the file in the primary filegroup. The File Name that I added is Crawl.ndf; the ndf extension is commonly used for additional database files. Finally make sure to give some thought to the Initial Size and Autogrowth properties based on your environment.

If you like to script out these kinds of database changes, here is the script that matches what we just did above.

```
USE [master]
GO
ALTER DATABASE [SharedServices1_Search_DB]
ADD FILEGROUP [CrawlFileGroup]
GO
ALTER DATABASE [SharedServices1_Search_DB]
ADD FILE (
  NAME = N'Crawl',
  FILENAME = N'F:\DATA\Crawl.ndf' ,
  SIZE = 10240KB ,
  FILEGROWTH = 1024KB
) TO FILEGROUP [CrawlFileGroup]
GO
```

Move the Crawl Tables and Indexes to the New Filegroup

When you create a database object (e.g. table, index, etc.) you can specify the filegroup where you want to store the object. This is done with the ON clause as shown in the sample script below:

```
CREATE TABLE dbo.SampleTable (
 sample_text nvarchar(256)
) ON CrawlFileGroup
```

If you do not specify the ON clause, the object will be created in the default filegroup. In our earlier screen shot that shows creating a new filegroup, you can see that the PRIMARY filegroup has the Default checkbox checked.

When you want to move objects from one filegroup to another, you essentially have to recreate them. The following script creates a general-purpose stored procedure to move a table to a filegroup (this script was taken from this blog post; I have not changed it other than adding comments and reformatting):

```
CREATE PROCEDURE [dbo].[proc_MoveTableToFileGroup]
(
 @fileGroup sysname,
 @tableName sysname
)
as
begin
  declare @data_space_id int
  declare @object_id int
  declare @index_id int
  declare @index_name sysname
  declare @object_name sysname
  declare @fileGroupName sysname
  declare @index_cols nvarchar(4000)
  declare @sql nvarchar(4000)
  declare @key_ordinal int
  set @index_id = 0
  set @key_ordinal = 0
  --- STEP 1: validate filegroup exists
  select
   @data_space_id = data_space_id
  ,@fileGroupName = name
  from sys.filegroups
  where name = @fileGroup

  if @data_space_id is null
  begin
    raiserror ('The specified filegroup does not exist.', 16, 1)
    return
  end
  -- STEP 2: iterate over each index for table
  while 1=1
  begin
    -- STEP 3: get the next index for the table
 select top 1
  @object_id = i.object_id
 ,@index_id = i.index_id
 ,@index_name = i.name
 ,@object_name = o.name
 from sys.indexes AS i
 inner join sys.objects AS o
  ON i.object_id = o.object_id
 where i.index_id > 0
   and o.type = 'U'
   and o.name = @tableName
   and i.index_id > @index_id
   and i.data_space_id <> @data_space_id
 order by i.index_id

 -- STEP 4: exit if not index found
 if @@rowcount = 0
   begin
    if @index_id = 0
      print 'There are no indexes to move to filegroup ' +
  @fileGroupName + ' for table ' + @tableName
      break
   end

 set @index_cols = null
 set @key_ordinal = 0
```

```
-- STEP 5: construct index column list
while 1=1
begin
  select top 1
   @index_cols = case when @index_cols is null
     then '[' + c.name + ']'
  else @index_cols + ', [' + c.name + ']' end +
 case when i.is_descending_key = 0
  then ' asc' else 'desc' end
  ,@key_ordinal = i.key_ordinal
  from sys.index_columns i
  inner join sys.columns as c
   on i.object_id = c.object_id
   and i.column_id = c.column_id
  where i.object_id = @object_id
  and i.index_id = @index_id
  and i.key_ordinal > @key_ordinal
  order by i.key_ordinal
  if @@rowcount = 0
   break
end
-- STEP 6: dynamic sql statement
select @sql =
 -- STEP 7: drop/add primary key
 case when i.is_primary_key = 1
  then
   N'begin try ' +
   N'begin tran ' +
   N'alter table [' + @object_name +
     '] drop constraint [' +
     i.name + '] ' +
   N'alter table [' + @object_name +
     '] add constraint [' + i.name + '] ' +
     'primary key ' +
  case when  i.type = 1
   then 'clustered ' else 'nonclustered ' end +
  ' (' + @index_cols + ') ' +
  'with (' + 'IGNORE_DUP_KEY = ' +
  case when  i.ignore_dup_key = 1
   then 'ON ' else 'OFF ' end +
  ', PAD_INDEX = ' +
   case when  i.is_padded = 1
    then 'ON ' else 'OFF ' end +
   case when  i.fill_factor = 0
    then '' else ', FILLFACTOR = ' +
    CONVERT(nvarchar(10),i.fill_factor) end +
   ') ' +
  'ON [' + @fileGroupName + ']' +
  N' commit ' +
  N'end try ' +
  N'begin catch ' +
  N'rollback ' +
  N'end catch '
  else
   -- STEP 8: create index with drop existing
   N'create ' +
      case when  i.is_unique = 1
```

```
        then 'unique ' else '' end +
    case when  i.type = 1
     then 'clustered ' else 'nonclustered ' end +
    'index [' + i.name + '] on [' + @object_name +
    '] (' + @index_cols + ') ' +
    'with (' +
    'IGNORE_DUP_KEY = ' +
    case when  i.ignore_dup_key = 1
     then 'ON ' else 'OFF ' end +
    ', PAD_INDEX = ' +
    case when  i.is_padded = 1
     then 'ON ' else 'OFF ' end +
    case when  i.fill_factor = 0
     then '' else ', FILLFACTOR = ' +
      CONVERT(nvarchar(10),i.fill_factor) end +
    ', DROP_EXISTING = ON ) ' +
    'ON [' + @fileGroupName + ']'
  end
 from sys.indexes AS i
 where i.object_id = @object_id
 and i.index_id = @index_id
 print 'Moving index ' + @index_name +
  ' to filegroup ' + @fileGroupName
 print @sql
 print ''

 -- STEP 9: execute dynamic sql
 exec sp_executesql @sql
 end
end
```

The following are the main points about the stored procedure:

- Create the stored procedure in your search database; i.e. USE your-search-database then execute the CREATE PROCEDURE script.
- The stored procedure takes two parameters: the destination filegroup name and the table name to be moved to that filegroup.
- STEP 1 checks to see whether the filegroup passed in as a parameter does in fact exist in the database; an error is raised if it doesn't and the procedure exits.
- STEP 2 is a while loop that iterates over all of the indexes for the table passed in as a parameter.
- STEP 3 gets the first or next index for the table. Note that the select statement will only return 1 row (or no rows) and each column is assigned to a variable for use elsewhere in the stored procedure. There are a number of interesting things to point out in the where clause:
  - An index_id of 0 is an entry for a table that does not have a clustered index; any such table is filtered out of the results.
  - The table name parameter is used to return only the indexes for the table.
  - The filter i.index_id > @index_id makes sure that each index for the table is only selected once.
  - The filter i.data_space_id <> @data_space_id ignores any index that is already located on the destination filegroup.
- STEP 4 checks if there are no indexes (or no more indexes) for the table; if there are no indexes for the table (i.e. @index_id = 0) then a break statement is executed which ends the while loop.
- STEP 5 retrieves all of the column names for the index selected in STEP 3 (in ordinal number order) and stores them in the @index_cols variable. The column order (ascending or descending) is added. This will be used later in the dynamic SQL created in STEP 6.
- STEP 6 creates a dynamic SQL statement that will be executed at the end of the stored procedure.

- STEP 7 creates the dynamic SQL to drop a primary key constraint and recreate it. The key point is the use of the ON clause which specifies the destination filegroup.
- STEP 8 creates the dynamic SQL for any index which is not in support of a primary key. Note the use of the ON clause to create the index in the destination filegroup and also DROP_EXISTING = ON which drops the existing index (required because the index already exists).
- STEP 9 executes the dynamic SQL statement created in STEP 7 or STEP 8.

After creating the stored procedure, execute the following script to move the crawl tables to the destination filegroup. This script was taken from this blog post; I have not changed it other than reformatting:

```
declare @fileGroup sysname
set @fileGroup = 'CrawlFileGroup'
if not exists (
  select 1
  from sys.filegroups
  where name = @fileGroup
)
begin
  raiserror ('The specified filegroup does not exist.', 16, 1)
  return
end
exec proc_MoveTableToFileGroup @fileGroup, 'MSSAnchorChangeLog'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSAnchorPendingChangeLog'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSAnchorText'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSAnchorTransactions'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlChangedSourceDocs'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlChangedTargetDocs'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlContent'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlDeletedErrorList'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlDeletedURL'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlErrorList'
begin try
  begin tran
  IF  EXISTS (
    SELECT *
    FROM sys.foreign_keys
    WHERE object_id =
    OBJECT_ID(N'[dbo].[FK_MSSCrawlContent_MSSCrawlHistory]')
    AND parent_object_id =
    OBJECT_ID(N'[dbo].[MSSCrawlContent]')
  )
  ALTER TABLE [dbo].[MSSCrawlContent]
  DROP CONSTRAINT [FK_MSSCrawlContent_MSSCrawlHistory]
  exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlHistory'
  ALTER TABLE [dbo].[MSSCrawlContent]
  WITH CHECK ADD  CONSTRAINT [FK_MSSCrawlContent_MSSCrawlHistory]
  FOREIGN KEY([CrawlID])
  REFERENCES [dbo].[MSSCrawlHistory] ([CrawlID])
  commit
end try
begin catch
  rollback
end catch
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlHostList'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlQueue'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlURL'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSCrawlURLLog'
exec proc_MoveTableToFileGroup @fileGroup, 'MSSTranTempTable0'
```

The main points about the above script are:

- Execute the script in your search database; i.e. use your-search-database
- It checks to see that the destination filegroup exists.
- The bulk of the script just executes the proc_MoveTableToFileGroup stored procedure (discussed earlier) once for each of the crawl tables.
- There is one foreign key constraint that must be dropped then recreated as shown above.

When you run the above script you will see messages output for each table as it is processed.

Caveats

These caveats are noted in the blog post on the Microsoft Enterprise Search Blog:

- This may take quite a long time to complete, depending on the size of the tables.
- The built-in SharePoint backup and restore utilities are not filegroup aware; i.e. if you do a restore then the same drives must exist as when you performed the backup.
- It is not recommended that you use SQL Server backup and restore on the search database as some of the data used for search is not in the search database (it exists on the file system).
- Any SharePoint update that you install (hot fix, service pack, etc.) you can use proc_MoveTableToFileGroup stored procedure and/or change an index and move it back to the primary filegroup. You should plan on rerunning the above steps after such an update. The stored procedure doesn't touch any indexes that are already on the destination filegroup.

Next Steps

- This tip is a good example of some very specific Microsoft guidance with respect to a SharePoint database.
- You have to poke around the TechNet website in order to find these types of things; they are often links that you find after following other links.