

Data Storage with SQL Server

The storage engine represents the heart of a SQL Server. No matter what front end accesses the data – glitzy Web sites or on-line transaction processing (OLTP) systems – data will need to be retrieved, added or amended in the database engine. The other dead cert is that your database will change and grow over time – sometimes radically, and sometimes beyond your wildest planning.

Building a storage facility capable of scaling in line with these requirements is a tough job. User expectations have grown out of all recognition over the past few years, especially in the current climate when business demands more functionality from less investment.

SQL Server Storage Architecture

There are two main components to the SQL Server 200 relational database, the storage engine and the relational engine. Although they are independent from each other they obviously interact and do so using interfaces such as OLE DB.

The storage engine has a number of duties including:

- Recovering the database from system failure
- Management of files and database pages used to store data
- Manage data buffers and system IO to the physical data pages
- Manage locking and concurrency issues

A lot of storage engine functionality happens automatically – that is the DBA does not need to regularly intervene to manage routine tasks and the database will endeavour to fine tune itself. In fact the chances are that the database will do a much better job than the average DBA, in most circumstances.

Data flow through the storage engine will follow a number of logical steps. The relational engine is responsible for compiling and optimising the submitted T-SQL statement into an execution plan which breaks the statement into a series of logical steps to access the data from tables and indexes. The steps are then executed by the relational engine calling data from the storage engine. The returned data is combined by the relational data into a final result set and sent back to the original client. To speed up queries SQL Server 2000 introduced a process so that appropriate query predicates are sent to the storage engine ahead of time delivering some fairly considerable performance gains.

When managing the storage engine SQL Server needs to balance resources with the operating system.

The easiest configuration is a single server dedicated to SQL Server. The database will then use all possible resources selfishly – memory and processor. This is especially useful when running large index builds or database processes.

The memory pool for SQL Server can be quite large – up to 64GB for SQL Server 2000 Enterprise Edition. SQL Server will use this for a range of purposes including caches which can be used to store recently accessed database tables or stored procedure code. This memory management is “hands off” as far as the DBA is concerned as the database will dynamically allocate and de-allocate memory as required. The only time that this will need specific intervention is if you are running multiple SQL Server instances on a single server or allowing SQL Server to sit on a server with another application. Be warned though, running SQL Server on the same machine as a Windows 2000 domain controller or Exchange Server is not a good idea unless you have lots of memory and processors – even then it might not be wise.

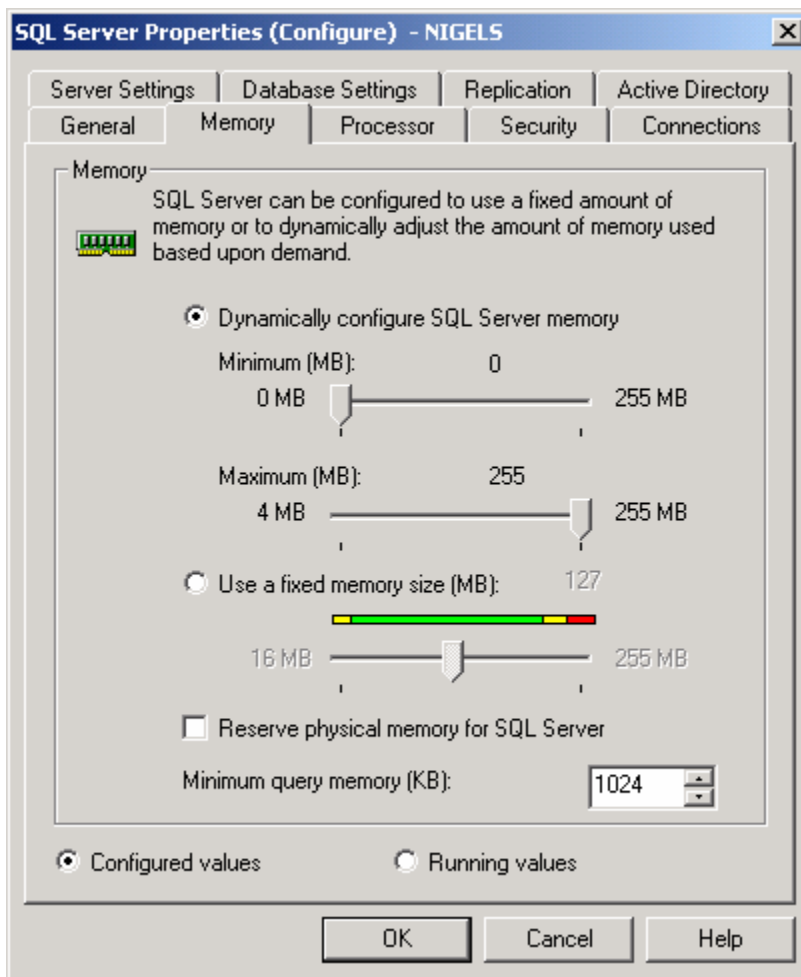


Fig. Memory setting on a very small server – my laptop!

This access to a large memory pool also helps with a significant bottleneck in the storage engine –access to disk resource, better known as file I/O. Each time data

is read from or written to disk then a cost is incurred in the performance of the database. By writing data to memory the read/write time is significantly reduced.

Physical Data Storage in the Database

The basic storage unit of SQL Server is the page which can store up to 8KB of data, producing 128 pages per megabyte. Unfortunately not all this 8KB is available to you for data storage as each page needs to carry 96bytes of header information to store information such as free space on the page and the object ID of the owning object (for example a table).

There are 8 different types of page used in SQL Server:

- Data – contains data except text, ntext and image data
- Index – index entries
- Text/Image - contains text, ntext and image data
- Page Free Space – details on how much free space is available on the pages
- Global Allocation Map, Secondary Global Allocation Map (GAM/SGAM) – data about extent allocation
- Index Allocation Map – details on any extents used by a table or index
- Bulk Changed Map – details about extents modified by any bulk operations since the last BACKUP LOG statement
- Differential Changed Map – Information on how any extents may have been changed since the last BACKUP DATABASE

Data rows are placed on the page immediately after the header information. There is a row offset table with an entry for each row on the page containing a value for how far the first byte is from the start of the page. This sits at the end of the page and the table is in reverse sequence from the sequence of the rows in the page.

The storage engine does not allow rows to span multiple pages.

Extents are a set of 8 contiguous pages or 64KB or 16 extents per MB. For improved space management tables with small amounts of data don't get allocated entire extents. Uniform extents are owned by a single database object and all of the 8 pages can only be used by the owning object. Mixed extents can be used by up to eight objects. A new table or index is usually allocated to a mixed extent but when it grows to eight pages will be switched to a uniform extent.

Pages and extents manifest themselves as physical files.

Physical File Structure used by the Storage Engine

A database will typically be mapped over a number of operating system files. Files can only be used by one database and data and log information is never mixed in the same files.

A database will start with a primary file which in turn points to any subsequent files used by a database. Conventionally these files have .mdf as their filename extension. Secondary data files then contain all of the other associated data for a database and come with the extension .ndf. Log files are used to recover the database and there will always be at least one log file per database with the extension .ldf. These file extensions are only conventions and can be customised, but it is not recommended as it may cause you some support issues later on.

Database Files

The pages in a SQL Server file are numbered from 0 upwards sequentially. Files have their own unique ID number so the file number and page number need to be given to locate a specific page. The first page in a file is a file header page that contains details about the make up of the file. Other pages at the start of the file have system based information. Unlike much earlier versions of SQL Server these files can grow automatically, the growth factor being set by a specific growth increment. If there are a group of files in a file group then all of the files will need to be full before the file sizes autogrow using a “round robin” algorithm. Naturally not even SQL Server can permit growth of files beyond the physical disk limit so ensure you have enough room to spare.

A file group is a number of database files brought together for easier allocation and administration. This has been used by some DBAs to place specific types or ranges of data onto specific physical hard disks to really squeeze some extra performance out of the database. Membership of a file group is an exclusive process – once you are in a file group you cannot belong to another without leaving the original group. Log data is never managed as part of a file group as it has a different role to the other data file types.

File groups fall into two categories – primary and user defined. The primary file group contains the primary data file and every other file not explicitly assigned to a file group including all of the system table pages. User defined groups have been set up by the DBA for specific storage reasons.

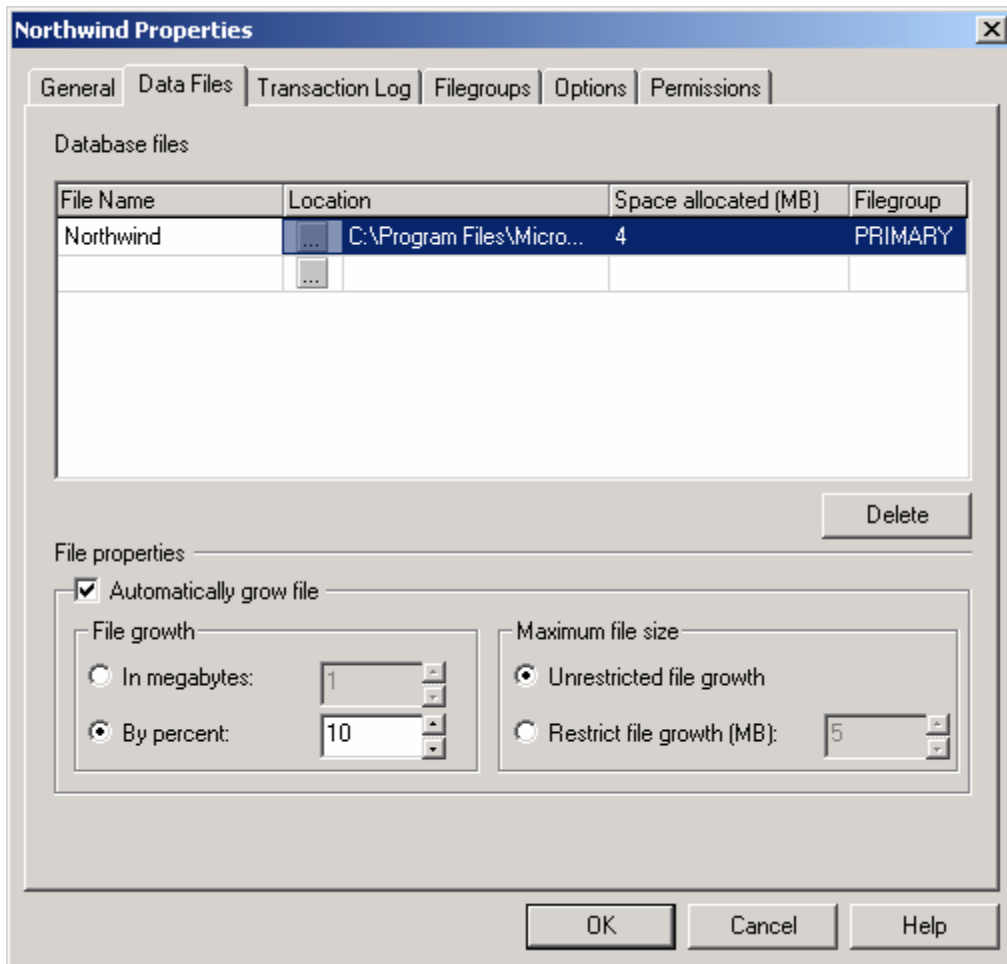


Fig. Data files for Northwind Database

Managing Space

Once data starts to fill the database how is space managed for best effect? The objective of space management algorithms is to ensure that any free space information is easy to get to, so reducing the number of reads necessary to work out where data should be placed. Any data allocation information should be chain free, so that simple updates don't require complex chains of updates to force through allocation data.

This is facilitated by the Global Allocation Map (GAM) and Shared Global Allocation Map (SGAM). The GAM keeps track of what extents have been allocated using a binary 1 or 0 indicating if an extent is free or allocated. The SGAM tracks the use of mixed extents using 1 to indicate if the extent is mixed and has free pages or 0 if it is not a mixed extent or it is a mixed extent but with no free pages.

SQL Server 2000 uses state of the art storage techniques to ensure that data is stored as effectively as possible. It will be interesting to see what changes to this we will see in the next version of SQL server, "Yukon".