DBA Handbooks

Brad's Sure Guide to SQL Server Maintenance Plans

Brad McGehee

ISBN: 978-1-906434-33-5

Brad's Sure Guide to SQL Server Maintenance Plans

By Brad M. McGehee

First published by Simple Talk Publishing 2009

ISBN 978-1-906434-33-5

Cover Image by Paul Vlaar
Edited by Tony Davis
Typeset & Designed by Matthew Tye & Gower Associates
Copy Edited by Gower Associates

Table of Contents

About the Author

Brad McGehee, currently Director of DBA Education at Red Gate Software, is a SQL Server DBA, trainer and writer with over 15 years' SQL Server experience, and over 6 years' training experience. He is an accomplished Microsoft SQL Server MVP, and was the founder of the popular community site SQL-Server-Performance.Com, which he operated from 2000 through 2006, writing over one million words on SQL Server topics.

Brad is a frequent speaker at SQL PASS, European PASS, SQL Connections, SQLTeach, devLINK, SQLBits, SQL Saturdays, TechFests, Code Camps, SQL Server user groups, and other industry seminars. In 2009, Brad made 33 public presentations to a total of 1,853 attendees, in six different countries.

A well-respected and trusted name in SQL Server literature, Brad is the author or co-author of more than 15 technical books and over 100 published articles. His most recent books include *How to Become an Exceptional DBA (2nd Edition)*, *Brad's Sure Guide to SQL Server 2008*, and *Mastering SQL Server Profiler*, all of which are available in PDF format at: http://www.sqlservercentral.com/Books/

When he is not travelling to spread his knowledge of SQL Server, Brad enjoys spending time with his wife and young daughter in Hawaii.

Acknowledgements

I want to thank my wife, Veronica, and my daughter, Anna, for their support while I wrote this book.

I also want to thank Tony Davis, my editor, for making me look good in print.

Introduction

SQL Server has a reputation as being a simple database application to install, configure, and maintain. This is a little misleading. SQL Server is a powerful relational database that can handle the needs of the largest organizations and, as such, its proper maintenance almost certainly requires the attention of an experienced DBA.

This reputation, coupled with the fact that it is relatively inexpensive, means that SQL Server has become a favorite platform for multiuser applications, and it often appears in organizations who cannot afford to have experienced DBAs on their staff. In many cases, organizations have SQL Server instances that are maintained by a part-time DBA, or an "accidental DBA," who may be a network administrator, developer, accountant, or even an office clerk. In the worst cases, nobody is looking after the health of the SQL Servers.

Millions of SQL Server instances run in the offices of small and medium-sized organizations, more than the total number of instances that run in large organizations, and so it follows that there are many accidental DBAs out there, who often don't have the knowledge, the experience, or the time to perform the appropriate level of maintenance on their SQL Server databases, much as they might like to. This can mean poor performance and reduced availability.

Although not a perfect solution to this problem, SQL Server does offer two closely-related tools that make it easier for part-time, non-professional DBAs to perform at least the "required minimum" level of maintenance on their SQL Server instances.

These two tools are:

- **Maintenance Plan Wizard** – a Wizard that steps the user through the process of setting up basic Maintenance Plans, with limited options.

- **Maintenance Plan Designer** – a drag-and-drop GUI interface in SSMS that facilitates the design and creation of more flexible, customizable maintenance plans.

Unfortunately, neither tool is especially easy to use or well documented. However, with the guidance I hope to provide in this book, they can become powerful tools in helping the "accidental DBA" to perform critical maintenance tasks, and so help to ensure SQL Server's performance and availability. In addition to learning how to use these tools you will, along the way, pick up a lot of good, general advice on SQL Server database maintenance.

Who Should Read this Book

This book is targeted at the following groups of DBAs.

- **Accidental/involuntary DBAs**, who fell into the role of DBA "by accident" and who don't have a strong background in database administration.
- **Part-time DBAs**, whose DBAs tasks are only a small part of their regular job duties, and whose DBA skills range from novice to intermediate.
- **Full-time DBAs**, who are at the novice to intermediate level in terms of their knowledge and experience.

If you fall into one or more of the above categories, then this book is for you, as it will not only explain what database maintenance needs to be done, but how to do it properly using the Maintenance Plan Wizard and/or the Maintenance Plan Designer.

More generally, I would suggest that these tools are most suitable for DBAs who:

- are not T-SQL or PowerShell experts, but who are able to get around in SQL Server Management Studio (SSMS)
- typically have 25 or fewer SQL Server instances to manage
- typically have databases that are less than 100 GB
- are probably using the Standard Edition of SQL Server
- have an available maintenance window on a daily or weekly basis (24/7 uptime is not a requirement).

If, on the other hand, you are an experienced DBA, managing many SQL Server instances, or very large databases, or lots of simultaneous users, or needing 24/7 uptime, then these tools are probably not, in general, suitable for your requirements. In fact, you're probably already using custom T-SQL or PowerShell scripts to perform your database maintenance.

Having said this, although they are sometimes reluctant to admit it, I know many experienced DBAs who still use the Maintenance Plan Wizard and/or the Maintenance Plan Designer from time to time. Alongside their "mission critical" systems, even experienced DBAs still maintain the databases of smaller, less active SQL Server instances and, for this purpose, these tools are the quickest and easiest way to create and schedule the set of maintenance tasks that will help ensure the continued smooth running of these systems.

Goals of this Book

As I cover how to use the Maintenance Plan Wizard and Maintenance Plan Designer in this book, I have tried to keep the following goals in mind:

- to keep the book at a level that most non-professional DBAs can understand
- not only to cover the mechanics of how to use the Maintenance Plan Wizard and Maintenance Plan Designer, but also to offer practical advice on how best to maintain your databases
- to provide an easy-to-read, tutorial approach to learning
- to offer lots of best practices from the real world.

SQL Server Editions Covered in this Book

This book covers the use of the Maintenance Plan Wizard and the Maintenance Plan Designer for SQL Server 2005 and SQL Server 2008, including both the Standard and Enterprise editions. If you are running SQL Server 2005, you should be on Service Pack 2 or later, as Service Pack 2 introduced some changes in the Maintenance Plan Wizard and Maintenance Plan Designer which make it closer in functionality to SQL Server 2008.

All the screenshots and examples are from SQL Server 2008, which, on occasion, varies from SQL Server 2005. When there are significant differences, I will point them out.

SQL Server 2000 and earlier is not covered because Maintenance Plans changed substantially between SQL Server 2000 and SQL Server 2005. Although the implementation changed quite a bit, the database maintenance recommendations I make in this book still apply to SQL Server 2000 and earlier.

Chapter 1: Why is Database Maintenance Important?

More times than I can count, I have seen a company install SQL Server databases without first creating any form of maintenance plan. These servers hum merrily along with nary a problem. That is, until there *is* a problem. At this point, query performance drops drastically or servers run out of disk space or, in extreme cases, databases become corrupt. And oh, by the way, nobody ever bothered to set up a backup plan, so there are no backups to restore. Oops!

The goal of implementing a database maintenance plan is to help prevent the kinds of problems just described. If implemented correctly, a database maintenance plan can help ensure that a SQL Server's databases perform adequately and, if there should be a problem, provide the necessary backups to minimize the loss of any data. Another benefit of implementing a database maintenance plan is that it helps to prevent, or to catch early, many different kinds of database-related problems. By being proactive with a good maintenance plan, time spent troubleshooting problems after the fact is often reduced.

In this chapter, we'll review some of the most important database maintenance tasks with which a DBA must be concerned, such as database backups and integrity checks, which will be included in virtually every database maintenance plan.

We'll then consider the four major tools available to implement these maintenance tasks. We'll focus on the two tools that are at the heart of this book, namely the database **Maintenance Plan Wizard** and the **Maintenance Plan Designer**, but we will also consider the options of using T-SQL scripting and PowerShell scripting.

The Scope of Database Maintenance

If you were to ask ten different DBAs to define "database maintenance," you would probably get ten different answers. The problem is that the term "database maintenance" is not clearly defined within the DBA community. Taken literally, the term refers to the maintenance of SQL Server *databases*. However, most DBAs confer on the term a more general meaning, encompassing maintenance of not only the databases, but also the SQL Server instances on which they reside, the OS, and the physical box on which SQL Server runs.

Every part of the larger SQL Server environment needs to be carefully managed and maintained in order to assure a high level of performance and availability. However, for the

purposes of this book, I am going to interpret the term quite literally, and define it as follows:

> ***Definition: Database maintenance plan***
>
> *A database maintenance plan is a set of specific, proactive tasks that need to be performed regularly on* **databases** *to ensure their adequate performance and availability.*

In other words, this book focuses solely on databases and on how to use the Maintenance Plan Wizard and the Maintenance Plan Designer to do basic database maintenance. Important as they are, this book does not cover other issues surrounding the health of the broader SQL Server ecosystem. As such, while everything in this book is important, it is only a subset of all the things that a DBA needs to do to maintain healthy SQL Servers. For more information on these broader topics, do an Internet search on "SQL Server Best Practices" to find additional information.

My goal in this book, indeed the goal of the Maintenance Plan Wizard and Designer, is to cover those critical database maintenance tasks that, as a bare minimum, should be applied to all databases, to ensure *adequate* performance and availability. Is "adequate" as opposed to "optimal" performance good enough? This, ultimately, is a business decision, based on the nature of the business function that a given database supports, and on the amount of time, resources, and money that the organization is prepared to invest. If an organization doesn't have the resources (or isn't willing to expend them) then, up to a point, it has to accept slower performance and lower availability from its SQL Servers.

This is a perfectly rational choice. Many SQL Server instances, especially those with small databases or a small number of users, often don't need to be "optimized to perfection" for performance, or even to be highly available. If a query takes 15 seconds to return a result, or if a database goes down for a couple of hours, or even a day, the organization will continue to function. In such cases, the Maintenance Plans covered in this book will suffice to ensure that the databases operate smoothly, and with acceptable performance. They will also be well suited to the main target audience of this book; namely accidental DBAs, or full-time DBAs who are just starting out, and who manage smaller non-mission-critical SQL Server installations.

The same argument does not hold for databases that support mission-critical business functions. In these cases, you will also need to invest time in creating more flexible and powerful maintenance plans, probably using T-SQL or PowerShell scripting, rather than using the Database Maintenance Wizard and Designer. Of course, organizations that choose to have highly performing and highly available SQL Servers have to make a large resource investment to attain this goal. There is no right or wrong maintenance plan; just different choices based on different needs.

Different Approaches to Database Maintenance

There are many different ways that DBAs can choose to perform database maintenance. In this section, we'll take a look at four of these options, including their pros and cons. This should allow you to determine which option is best suited to your particular needs.

As noted earlier, the focus of this book is on the first two of these tools: the Maintenance Plan Wizard and the Maintenance Plan Designer.

Maintenance Plan Wizard

The Maintenance Plan Wizard is one of two tools that SQL Server provides to create Maintenance Plans.

> ***A note on terminology***
>
> *SQL Server uses the term "Maintenance Plan" (note the capitalization) to refer to a database maintenance plan created using either the Maintenance Plan Wizard or the Maintenance Plan Designer.*

Under the covers, each Maintenance Plan takes the form of an **SSIS package**, which is then scheduled to run under one or more **SQL Server Agent** jobs, and will perform the various tasks that make up a database maintenance plan. We'll cover this in more detail in Chapter 3.

The goal of the Maintenance Plan Wizard is to guide you, step by step, through the creation of a Maintenance Plan, without the need to do any coding, thus making the whole process easy and quick. While the Wizard doesn't include every possible database maintenance feature or option, it does include the core database maintenance tasks that all DBAs should be performing on their SQL Servers. As such, it is often an appropriate tool for the part-time/accidental DBA, or even for full-time DBAs. For example, if the databases are small, the number of users is low, high server availability is not required, and there are available maintenance windows, then this tool is more than adequate in most cases.

It also has the following advantages:

- **The resulting Maintenance Plan can be modified and extended**, if necessary, using the Maintenance Plan Designer. Many DBAs use the Wizard to create their "base" Maintenance Plan, and then use the Designer to tweak it.

- **The tool includes an option to create Multiserver Maintenance Plans**, meaning that you can create Maintenance Plans for multiple servers in a single step. However, this feature is awkward to configure and has some backwards compatibility problems, so it may not work for all SQL Server environments. As such, I tend to avoid using it. The same feature is available in the Maintenance Plan Designer and is discussed briefly in Chapter 16 (though it has the same drawbacks).

In many ways, the Maintenance Plan Wizard does attain its goal of easing the creation of database maintenance plans. However, it falls short in some areas, and can cause problems for the incautious. The Wizard assumes that you fully understand every option that it offers to you, and how each affects your databases. If you don't understand the options, and you guess at their meaning, it is very easy to create a Maintenance Plan that performs terribly. Unfortunately, the Wizard is not smart enough to prevent you making these poor choices. However, in this book, I will fully explain all these options so that you can use the tool to its full advantage, and avoid such performance issues.

As useful as the tool can be, DBAs must be fully aware of what it can and can't do. Having created a few Maintenance Plans with the Wizard, some novice DBAs confidently assume that that their databases are fully maintained. As we have already discussed, the Maintenance Plan Wizard only performs core maintenance tasks, rather than every possible database maintenance task that should be considered for a given database or server. For example, just because you create backups with the Wizard, this does not ensure that the backups are good (restorable), or that they have been moved off the server to protect them should the SQL Server instance experience a disk failure. Such tasks (other examples are covered a little later in this chapter) have to be done outside of the Maintenance Plan Wizard.

The Wizard also has the following specific shortcomings:

- **Limited number of database maintenance options**. If you need database maintenance options that are not provided, you'll have to resort to T-SQL or PowerShell scripts, or to use scripts for some tasks and the Wizard for others.

- **Lack of granularity**. For example, the Maintenance Plan Wizard can't determine which indexes need to be rebuilt, and which ones don't need to be rebuilt, and therefore has to rebuild them all. As such, it often takes more time to execute a Maintenance Plan created with the Wizard than a custom plan created using T-SQL or PowerShell scripts.

- **Inability to run multiple tasks**. Each type of maintenance task within a single Maintenance Plan can only be configured to run once within that Plan. This can make

some tasks more difficult than they need to be. For example, the maintenance task that is designed to delete older backup files can only delete one file type at a time, such as `BAK` or `TRN`, and not both at the same time. Because of this, you may have to create multiple Maintenance Plans just to perform simple tasks such as this.

- **No scripting to other instances**. Maintenance Plans created with the Wizard cannot be scripted and moved to other SQL Server instances, although multi-server Maintenance Plans can be created.

- **Bugs in some earlier versions of the Wizard**. If you use SQL Server 2005 Service Pack 2 or higher, or SQL Server 2008, then you should have no problems.

Some experienced DBAs will tell you that "real DBAs" don't use the Maintenance Plan Wizard and, instead, always write their database maintenance plans from scratch, using T-SQL or PowerShell scripts. In reality, this is not true. Many "real DBAs" use the Maintenance Plan Wizard, *when it is appropriate*. Much of this book will be devoted to letting you know when using the Maintenance Plan Wizard is appropriate, and when it is not.

Maintenance Plan Designer

If you search for the "Maintenance Plan Designer" in Books Online, you won't find anything referred to by this exact name. This is because I had to provide a name for a feature of SQL Server that does not appear to have a consistently-used, official name. Sometimes it is referred to as "New Maintenance Plan," or the "Maintenance Plan Design Tab," and other times as the "Maintenance Plan Designer Surface."

Essentially, the Maintenance Plan Designer is a drag-and-drop GUI interface found in SSMS, based on the SQL Server Integration Services (SSIS) Designer Surface, which allows DBAs to manually design and create Maintenance Plans from scratch, or to modify Maintenance Plans originally created using the Maintenance Plan Wizard.

The Maintenance Plan Designer offers more features than the Wizard and this, coupled with the element of manual control, means the DBA can create more comprehensive, flexible and customized Maintenance Plans than is possible with the Wizard.

NOTE

Chapters 16 to 19 cover the Maintenance Plan Designer in detail, after we've investigated the Maintenance Plan Wizard. The functionality offered by each tool overlaps substantially, so once you learn about the features of the Maintenance Plan Wizard, you will already know about most of the features of the Maintenance Plan Designer.

One advantage of the Designer over the Wizard, in my opinion, is that it shows you the T-SQL code that will be executed when a maintenance task runs. This code can help provide you with a better understanding of exactly what the task is doing, and can also be used as an example of how to use T-SQL to create your own maintenance plans, should you decide to write your own T-SQL code to enhance your Maintenance Plans. In addition, the Designer tool has the following specific advantages:

- **Control-of-flow ability**. The Designer allows you to create branching execution paths based on conditional logic. For example, you can specify that, if a particular maintenance task fails, then an e-mail is sent to the DBA team, notifying them of the problem.

- **Running multiple tasks**. Unlike the Wizard, you can run a task multiple times from within the same Maintenance Plan. This solves the problem described earlier with the Maintenance Plan Wizard. Now, within a single plan, you can delete both `BAK` and `TRN` files within a single Maintenance Plan.

- **Two additional tasks, only in the Designer.** An `Execute T-SQL Statement` task allows you to create a maintenance task that can do virtually anything, and have it run from within a Maintenance Plan. A `Notify Operator` task provides a powerful means to notify a DBA should a maintenance task fail to execute successfully.

Of course, the most obvious drawback of using the Designer is that it is a manual procedure and so is slower, and somewhat harder to learn than the Wizard.

Despite offering greater flexibility than the Wizard, the Designer still cannot match the power and flexibility of T-SQL and PowerShell scripts. In fact, aside from the ability to add conditional logic, the ability to run a task multiple times within a Plan, and the addition of two more tasks, the Designer suffers from most of the shortcomings listed for the Wizard.

Many DBAs might start off using the Maintenance Plan Wizard but, once they have mastered it, they often take the time to learn the additional features of the Maintenance Plan Designer, because the leap from learning the Wizard to the Designer is not a large one and, at the same time, they are gaining greater flexibility when creating Maintenance Plans.

T-SQL Scripts

Today, most full-time, experienced DBAs use T-SQL scripts, in combination with SQL Server Agent jobs, to perform their database maintenance. This is because T-SQL scripts offer 100% flexibility when it comes to database maintenance; you can do virtually anything you want or need to do.

For example, if you specify the `Rebuild Index` task in the Maintenance Plan Wizard, it will automatically rebuild all the indexes in a database. While this accomplishes the job of

rebuilding indexes, it is a resource-intensive process. The ideal solution is to run a script that identifies only the heavily fragmented indexes, and rebuilds them, but leaves the others alone, thus conserving server resources. Unfortunately, you can't do this with the Maintenance Plan Wizard; custom T-SQL or PowerShell scripts are required.

In addition, T-SQL scripts offer the following advantages:

- **OS access**. T-SQL offers the ability to access the Operating System (OS), although it is not always easy or as flexible as you might like. This is one option used by some DBAs to remove old BAK and TRN files.

- **Portability**. Appropriately written T-SQL scripts can easily be moved from server to server.

- **Script sharing**. Many DBAs share generic database maintenance T-SQL scripts on various community sites, so you don't have to reinvent the wheel. Of course, you don't want to run a script on your own server unless you fully understand what it does. You still need a good knowledge of T-SQL before using someone else's T-SQL script. Check out these URLs for examples of some freely availably T-SQL scripts used to perform database maintenance:

    - http://ola.hallengren.com/
    - http://sqlfool.com/2009/06/index-defrag-script-v30/
    - http://www.grics.qc.ca/YourSqlDba/index_en.shtml
    - http://www.simple-talk.com/sql/database-administration/sql-server-tacklebox-free-ebook/

Of course, all of this assumes a strong working knowledge of the T-SQL language, as well as a good understanding of SQL Server internals. For most people, this entails a long learning curve. Coding T-SQL scripts can be very time-consuming, and error prone. Sometimes debugging these scripts takes longer than writing them. In addition, if you are not careful about how you write your maintenance scripts, it is possible that when the next version of SQL Server is released your scripts may need to be modified (sometimes substantially) to work with the new version.

Finally, aside from third-party tools, there is no easy way to automate the execution of your T-SQL maintenance scripts across multiple servers. For that, you will need to learn PowerShell.

While T-SQL scripts might be the choice of most DBAs today, don't think this is the only option you have. If you want to keep database maintenance simple, then the Maintenance Plan Wizard and the Maintenance Plan Designer may work perfectly well. However, if you need an even more flexible option than T-SQL, consider using PowerShell scripts.

PowerShell Scripts

PowerShell is Microsoft's latest command-line shell scripting language that allows DBAs full access to the object models of both the OS and SQL Server. It also supports much more complex logic than T-SQL and has better error handling. This combination allows you to create extremely powerful and robust database maintenance scripts. PowerShell scripts, if written appropriately, can easily be used to perform database maintenance across multiple SQL Servers.

Microsoft has been avidly promoting PowerShell, although adoption has been slow, the main reason being that it involves learning a completely new object-oriented scripting language, which is very alien to many DBAs. On top of this, the DBA still needs to know T-SQL and SQL Server internals, as well as SQL Server Management Objects (SMO), and the OS Object Model (assuming you decide to take advantage of PowerShell's ability to access the OS).

This is a steep learning curve and means that PowerShell scripts, initially at least, can be even more time-consuming to write and debug than T-SQL. Also, whereas the appropriate T-SQL maintenance script can be run on most any SQL Server, many older servers may not have PowerShell installed.

As time passes, I am guessing that you will see more and more DBAs start moving from T-SQL scripts to PowerShell scripts, especially those who manage large numbers of SQL Server instances. This will continue to be a slow move, until more DBAs not only become familiar with the power and flexibility of PowerShell, but master the large body of knowledge needed to take full advantage of it.

In the meantime, the body of community scripts and knowledge is starting to grow. For examples of how to use PowerShell to perform database maintenance, check out this CodePlex.com project.

http://sqlpsx.codeplex.com/.

Alternatively, you can visit http://www.simple-talk.com and do a search for "powershell," to find many articles on the subject.

Core Maintenance Plan Tasks

As discussed earlier, the basic intent of the Maintenance Plan Wizard and Maintenance Plan Designer is to allow you to configure the "core" database maintenance tasks that must be performed on more or less every SQL Server database. These tasks are reviewed in the following sections.

Backup Databases

As obvious as this advice sounds, it is surprising how many SQL Servers I have run across that don't have proper backups. If your database becomes corrupt, and you don't have a restorable backup, then you will probably end up losing your data.

It is critical that any maintenance plan makes provision for the following two types of backup:

- **Full database backups** – backs up the data in the data (mdf) file(s) for that database. Full backups are the core of any disaster recovery plan.

- **Transaction log backups** – backs up the data in the log (ldf) file(s) for that database.

While most people understand why full database backups are important, some don't fully understand the rationale behind transaction log backups. The purpose of transaction log backups is twofold. Firstly, they serve to make a backup copy of all the transactions that have been recorded in the transaction log file since the last log backup. In the event of a disaster, these log backups can be applied to a restored copy of a full database backup, and any transactions that occurred after the full backup will be "rolled forward" to restore the data to a given point in time, and so minimize any data loss. For example, if you back up your transaction logs once an hour (and you have a valid full backup), then, theoretically, the most you could lose would be an hour's worth of transactions.

Secondly, for databases that use the full or bulk-logged recovery models, this action truncates the transaction log, so that it doesn't grow too large. Many part-time/accidental DBAs perform full backups on their databases, but they don't perform transaction log backups. As a result, the transaction log is not truncated, and it grows and grows until the drive it is on runs out of disk space, causing SQL Server to stop working.

It is the responsibility of every DBA to ensure that all appropriate databases are properly backed up and protected.

Verify the Integrity of a Database

It is possible for data in a SQL Server database to become corrupted, perhaps due to a failure in the disk subsystem, or some other event. While it is not common for a database to become physically damaged in this way, the possibility must be considered. Data corruption may occur only in one specific area of the database, and it's possible that the damage may not be discovered for some time, usually only when an attempt is made to query the corrupted data. Between the time at which the damage occurred and the time it was discovered, many days may have passed, and each of the backups made during this time will include the damaged data.

The longer the damage remains undiscovered, the more out of date will be the most recent undamaged backup. If you delete older backups on a regular schedule, you may not even have an undamaged copy! In either case, you may end up losing a lot of data, so it is important for DBAs to regularly check the physical integrity of their databases, using the `DBCC CHECKDB` command.

Maintain a Database's Indexes

Over time, as indexes are subjected to data modifications (`INSERT`s, `UPDATE`s, and `DELETE`s), index fragmentation can occur in the form of gaps in data pages that create wasted empty space, and in a logical ordering of the data that no longer matches the physical ordering of the data.

Both forms of fragmentation are normal byproducts of data modifications but, unfortunately, both can hurt SQL Server's performance. Wasted space reduces the number of rows that can be stored in SQL Server's data cache, which can lead to increased disk I/O. The index page ordering problem also causes extra disk activity, as it often takes more work to find the data on disk and move it to the data cache, than it would if the pages were in physical order.

SQL Server doesn't automatically correct index fragmentation problems. The only way to remove wasted space and restore the correct page ordering is to rebuild or reorganize the indexes on a regular basis. This requires the DBA to create a maintenance job to perform these tasks.

Maintain Index and Column Statistics

The Query Optimizer uses index and column statistics as part of its evaluation process, as it tries to determine an optimal query execution plan. If the statistics are old, or incomplete, then the Query Optimizer might create an inefficient execution plan, which substantially slows down a query's performance. In theory, index and column statistics are self-maintaining, but this self-maintaining process is not perfect in practice.

In order to ensure that the optimizer has the most complete and current statistics at its disposal, the DBA needs to create a maintenance task to ensure that they are regularly updated, either by rebuilding the indexes, or by updating the statistics using the `UPDATE STATISTICS` or `sp_updatestats` commands.

Remove Older Data from msdb

The SQL Server `msdb` database stores historical data about various activities, such as details about backups, SQL Server Agent jobs, and Maintenance Plan execution. If left unattended, the `msdb` database can grow over time to a considerable size, wasting disk space, and slowing down operations that use the `msdb` database. In most cases, this data does not need to be kept for a long period, and should be removed using such commands as `sp_delete_backuphistory`, `sp_purge_jobhistory`, and `sp_maintplan_delete_log`.

Remove Old Backups

While making database backups is important, you don't need to keep them for ever. If fact, if you don't clean up older backup files, your SQL Server's hard drives will quickly fill up, causing all sorts of problems. It is the job of the DBA to ensure that unneeded backups are removed from a SQL Server on a regular basis.

What's Outside the Scope of the Maintenance Plan Wizard and Designer?

While Maintenance Plans are a convenient way to perform much of your database maintenance work, neither the Wizard nor the Designer can do all your work for you. While the tasks included with Maintenance Plans are a good first start, the Wizard and designer aren't really intended to enable you to perform every single maintenance task that could be included in your database maintenance strategy.

For example, the following additional important database maintenance tasks are not covered by the Wizard or Designer:

- identifying and remove physical file fragmentation
- identifying missing, duplicate, or unused indexes
- protecting backups so that they are available when needed
- verifying that backups are good and can be restored

- monitoring performance
- monitoring SQL Server and operating system error messages
- monitoring remaining disk space
- and much, much more.

The moral of the story is that, while Maintenance Plans are a useful tool for many DBAs, they are not the perfect tool for all DBAs, and will only perform a subset of the required database maintenance tasks. If the Maintenance Plan Wizard or Designer meets your needs, then use them. On the other hand, if they don't properly meet your needs, then don't use them. Custom-created T-SQL or PowerShell scripts instead offer much more power and flexibility. While there may be a steep learning curve to create custom scripts, this is knowledge that you will be able to use elsewhere as a DBA, and it won't go to waste.

Summary

In this chapter, we have learned what database maintenance is, and why it is important, and we have considered the core database maintenance tasks that will comprise almost every database maintenance plan.

We also explored four different ways to perform database maintenance, including use of the Maintenance Plan Wizard and the Maintenance Plan Designer, which are the tools we'll focus on in this book, as well as T-SQL and PowerShell scripts.

In the following chapters, we learn how to create Maintenance Plans using the Maintenance Plan Wizard. As we learn how to use the Wizard, we will also be learning a lot of information that applies to the Maintenance Plan Designer, as both tools perform similar tasks and offer similar configuration options.

Chapter 2: Before you Create any Maintenance Plans...

Both the Maintenance Plan Wizard and the Maintenance Plan Designer have the ability to send e-mails to DBAs, providing them with information on the status of the Maintenance Plans that have executed. The number of e-mails and their contents vary depending on which tool you use to configure these e-mail notifications.

In the case of the Maintenance Plan Wizard, you can configure an option that will send you an e-mail every time that a Maintenance Plan has been executed, which includes standard information on what plan was run, and what steps were executed, and reports any errors that occurred. Once this option is configured, it will send an e-mail every time the Maintenance Plan is executed, and it will always include the same standard information.

The Maintenance Plan Designer, on the other hand, has more options. First, it allows the DBA to configure the conditions under which e-mail notifications will be sent. For example, instead of always sending an e-mail, as the Maintenance Plan Wizard does when a Maintenance Plan executes, the Maintenance Plan Designer can send e-mails based on conditional logic. So if you only want to receive e-mails if a Maintenance Plan fails, and not all the time, you can configure this. In addition, the Maintenance Plan Designer allows you to create custom e-mail messages, so that when you receive an e-mail, you know exactly what the problem is, instead of having to wade through a long report.

In either case, before you can receive e-mail messages from Maintenance Plans, created with either the Maintenance Plan Wizard or the Maintenance Plan Designer, there are two preliminary setup steps you must take:

1. Set up Database Mail.

2. Create one or more SQL Server Agent Operators, who will receive the e-mail notifications.

These two steps are described in detail, in this chapter. Having completed them, you'll be able to select the "e-mail report" option when creating a new Maintenance Plan. If you have existing Maintenance Plans that don't report via e-mail, you can modify them to do so using the Maintenance Plan Designer, as described in Chapter 19.

How to Configure Database Mail

While there are a couple of different ways to configure Database Mail, the easiest way is to use the Database Mail Configuration Wizard from within SSMS. To start this Wizard, navigate to the Management folder of the appropriate server, right-click on **Database Mail**, and select **Configure Database Mail**, as shown in Figure 2.1.

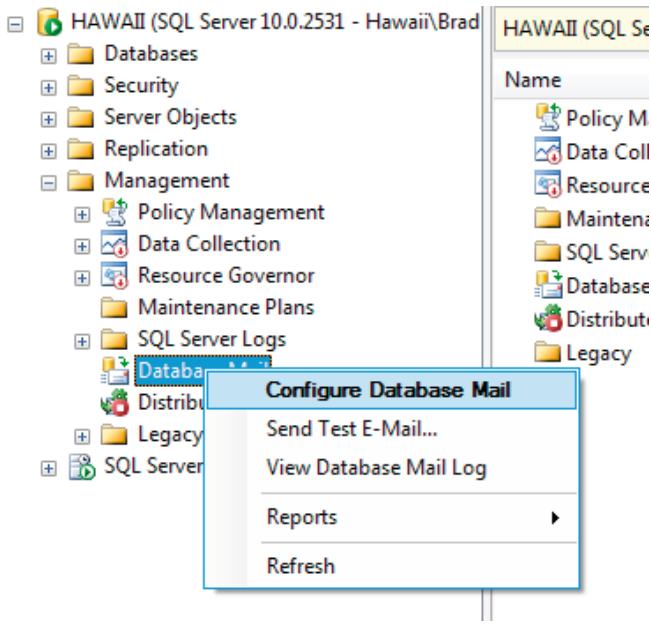

Figure 2.1: Using the Database Mail Configuration Wizard to set up Database Mail for the first time.

Click **Next** to get past the splash screen, and the Wizard starts off with the **Select Configuration Task** screen, as shown in Figure 2.2.

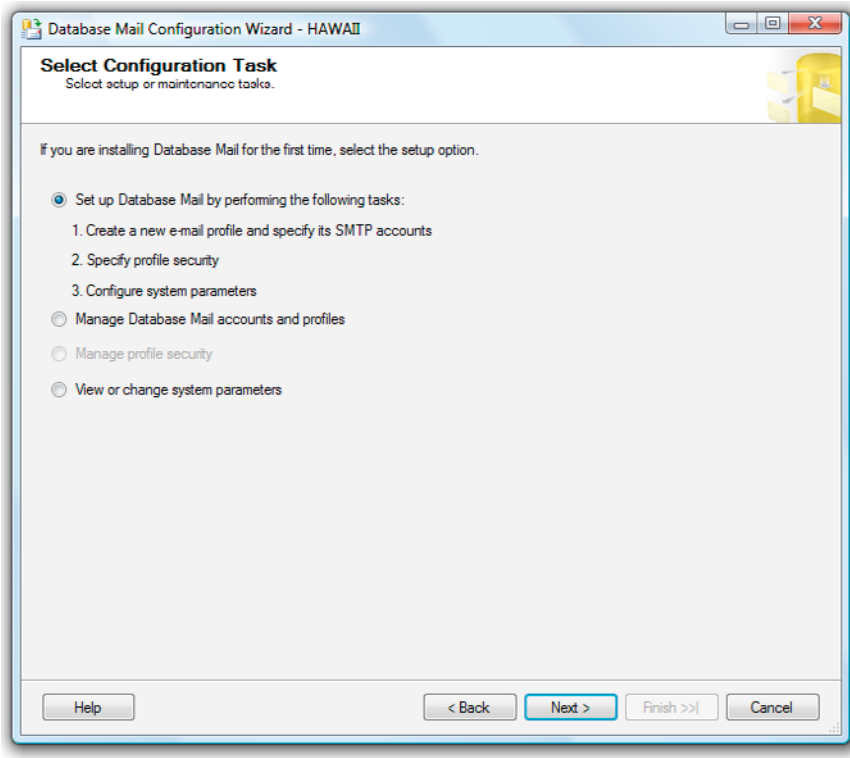

Figure 2.2: The Database Mail Configuration Wizard does several different tasks.

To set up Database Mail for use by the Maintenance Wizard, select the option **Set up Database Mail by performing the following tasks**. The wizard even lists the three tasks that need to be completed to set up Database Mail, which are explained shortly. Click on **Next** to continue.

If database mail has not yet been enabled for this SQL Server instance, then you will see the screen shown in Figure 2.3. If it has been enabled, you won't see the screen.

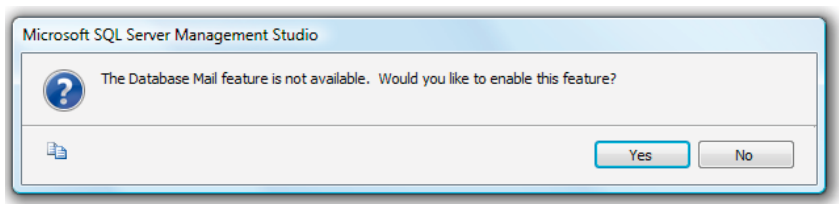

Figure 2.3: Before you can configure Database Mail, you must first enable it.

Assuming that Database Mail has not been turned on for this SQL Server instance, and you see the above screen, click **Yes**, and Database Mail will be enabled. The next screen, shown in Figure 2.4, will prompt you to create a **Database Mail profile**.

A profile is a collection of one or more SMTP accounts that can be used by SQL Server to send messages. In other words, when SQL Server wants to send a message, the message is sent to the profile, and then the profile is responsible for seeing that the e-mail is actually delivered. For fault tolerance, a profile can include more than one SMTP account. For example, if the profile tries to send an e-mail using one SMTP account, but it is not working, then the profile will attempt to send the e-mail using a second SMTP account, assuming one is available. Profiles can also be used as a security measure, either allowing or preventing someone from using it to send mail.

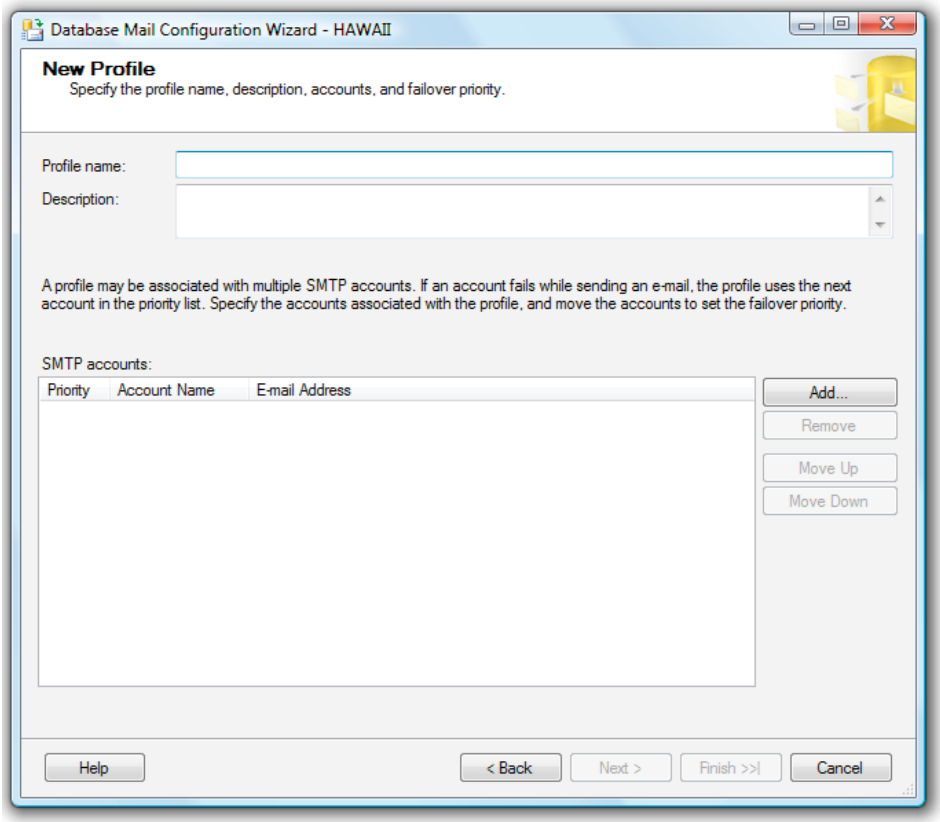

Figure 2.4: The first step when configuring Database Mail is to create a new profile.

To create a new profile, you must enter a profile name, an optional description, and then add and configure one or more SMTP accounts. In this example, we will create and configure a single SMTP account.

Multiple mail profiles

Database Mail can have multiple profiles to meet a wide variety of fault tolerance and security needs. Here, you just need a single profile for use by your Maintenance Plans.

Enter a descriptive Profile name, such as "Maintenance Plans." If your SQL Server instance has multiple mail profiles, then you'll probably want to enter a description of the intended use of this particular profile, so you don't get them confused.

Next, create and configure the SMTP account that will be used by Database Mail to send the e-mails from your Maintenance Plan. To create and configure a SMTP account, click on the **Add...** button, and the **New Database Mail Account** screen appears, as shown in Figure 2.5.

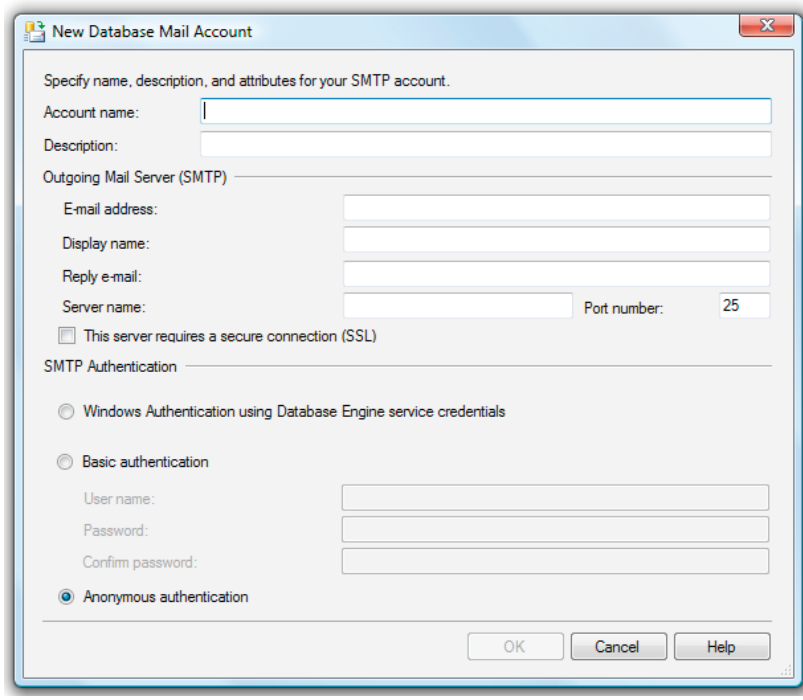

Figure 2.5: You will probably have to track down your SMTP server settings before you can complete this screen.

If you have never set up an e-mail client before, this screen might seem a little confusing. Essentially, you have to tell Database Mail what mail server it should use to deliver e-mail messages from your Maintenance Plans. If you're unsure, send the screenshot shown in Figure 2.5 to your organization's e-mail administrator, so he or she will know what SMTP settings you need.

When you ask the e-mail administrator for the SMTP settings, you also need to request that a special e-mail account be set up for use by SQL Server. For example, you might have an account set up called sqlserver@myorganization.com or sqlserveragent@... or maintenanceplan@... or some other descriptive name, so that when you receive an e-mail from Database Mail, you will know where it came from.

Let's take a look at each option, starting with the basic attributes of the SMTP account.

- **Account name**. The SMTP account must have a name so that it can be distinguished from other SMTP accounts used in the same profile. Since you are creating only a single SMTP account for your profile, what you call it is not very important. However, should you decide to have two SMTP accounts, for fault tolerance purposes, then you'd need to name them descriptively so that you can easily distinguish one from another. For example, you could use the name of the SMTP server as the account name, as that is a good way to distinguish one SMTP account from another.

- **Description**. This optional textbox can be left empty, or you can use it to help document your settings. For example, you might enter the name of the person who provided the SMTP settings, so you know who to go back to in the event of any problems.

- **Outgoing Mail Server (SMTP)**. This specifies attributes of the SMTP Server that will be sending the e-mail, including these six options:

  - **E-mail address** – the e-mail account that has been set up for use with SQL Server's database mail (for example, `sqlserver@myorganization.com`).

  - **Display name** – the display name of the above e-mail address that is shown as part of an e-mail. You will probably want to give it the same name as the user part of the e-mail address, such as "SQL Server," although you can use any name you choose that will remind you where the e-mail comes from.

  - **Reply e-mail** – the e-mail address used if someone should reply to an e-mail sent from the e-mail address entered above. Database Mail can't respond to e-mails it receives, so you can either leave this option empty, or add your own e-mail address, just in case someone should respond to an e-mail received from SQL Server.

  - **Server name** – the name of the SMTP mail server. It generally looks something like `mail.myorganization.com`.

- **Port number** – the port number used by your organization's SMTP server. E-mail servers communicate through specific TCP/IP ports, and the default port number, 25, is the one most commonly used, but it may not be the one your company uses, so be sure to check.

- **This server requires a secure connection (SSL)** – some SMTP servers require that SSL be turned on for additional security. Only select this option if you are instructed to do so.

The lower half of the screen is where we specify the SMTP Authentication options. In most cases, before an SMTP server will accept an e-mail, the sender must log on to the SMTP server with an authorized e-mail account. This is done to prevent spammers from accessing the SMTP server and using it. Find out which authentication model your organization uses, and complete the appropriate information, as follows:

- **Windows Authentication using Database Engine service credentials**: This option is not commonly used but, if it is, you have to ensure that the account used for the Database Engine service has permission to route mail to the SMTP server.

- **Basic authentication**: This is the most common method of authentication, and requires you to provide a user name and password. In most cases, the user name will be the e-mail address you entered above, and the password will be the password used for this e-mail address.

- **Anonymous authentication**: This option is rarely used because it allows anyone to access the SMTP server.

Having entered values for all of the settings, the screen will look similar to that shown in Figure 2.6.

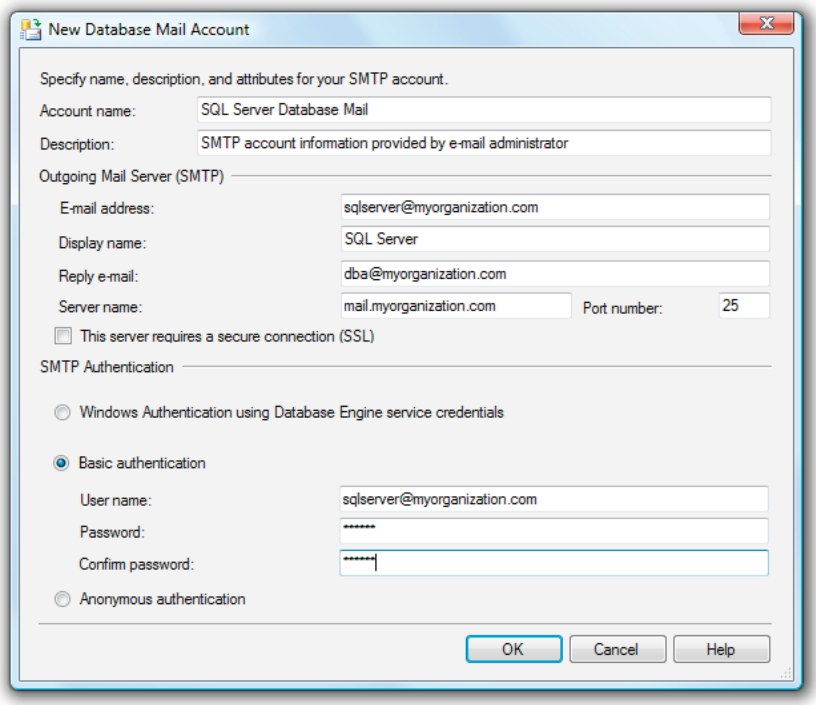

Figure 2.6: If the SMTP settings are not set correctly, Database Mail will not work.

Once you're happy with the account information, clicking **OK** will return you to the original mail Profile screen, which will display the SMTP account you just set up, as shown in Figure 2.7.

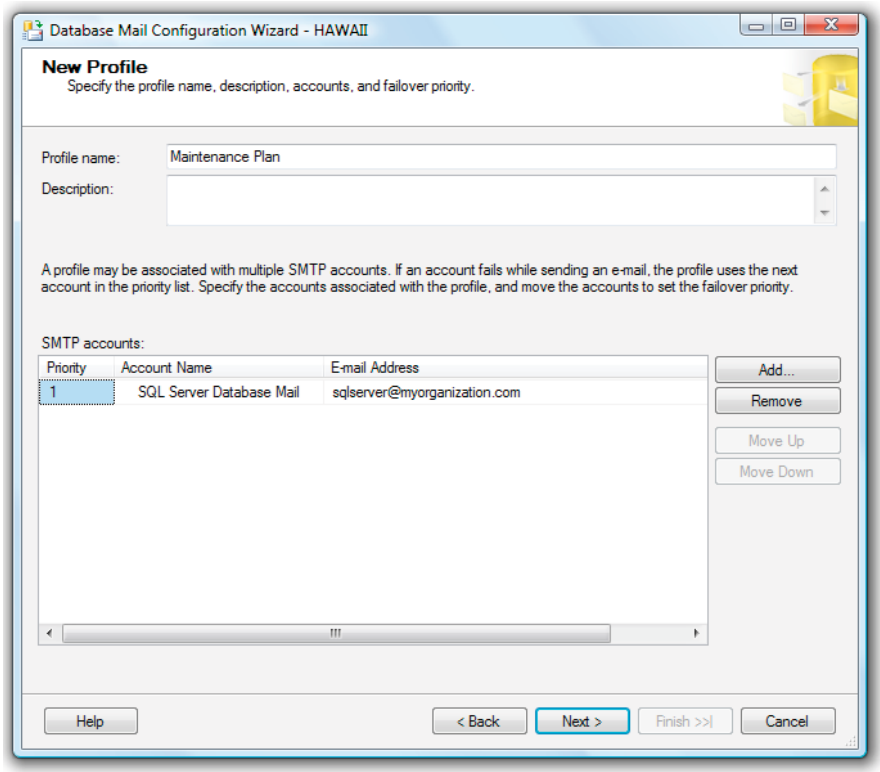

Figure 2.7: Although you are only setting up one account, you can see that multiple accounts can be set up if desired.

To continue with the Database Mail Configuration Wizard, click on **Next** to reach the **Manage Profile Security** screen, shown in Figure 2.8.

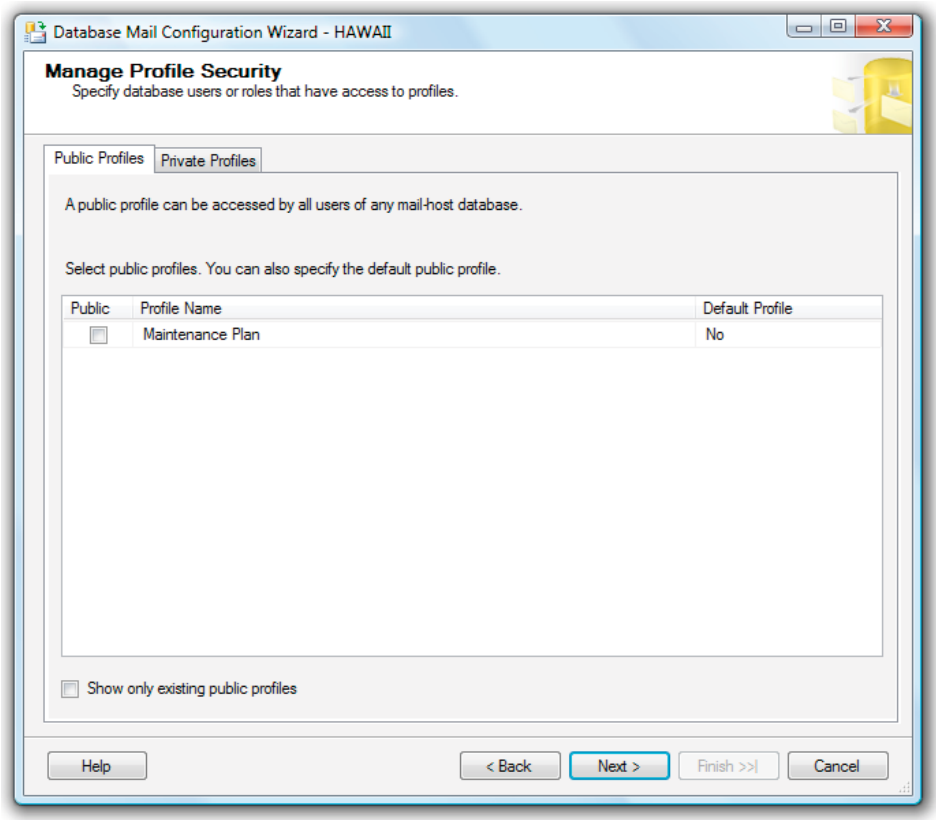

Figure 2.8: You have to specify whether a mail profile is public or private.

As shown in Figure 2.8, your Maintenance Plan profile has been created (it's called "Maintenance Plan"). Now you have to assign the profile as either **public** or **private**. A private profile is only usable by specific users or roles, while a public profile allows any user or role (with access to `msdb`) to send mail. To keep your setup as simple as possible, make the Maintenance Plan profile **public**, by selecting the checkbox under **Public** then clicking **Next** to move on to the **Configure System Parameters** screen, shown in Figure 2.9.

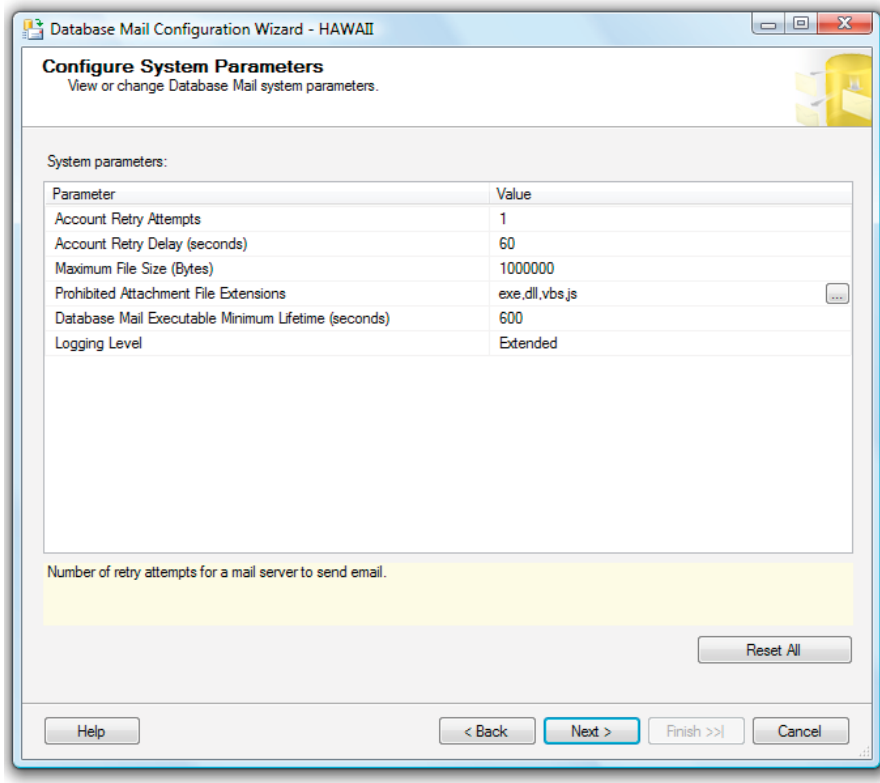

Figure 2.9: You have the opportunity to configure additional Database Mail parameters.

The last option in the Database Mail Configuration Wizard allows you to set the values of specific Database Mail parameters for the profile. Generally, we will leave these options at their default values. The only one I suggest you might consider changing is the value of the **Account Retry Attempts** parameter. By default, this value is 1, which means that there is only one attempt made to send an e-mail. If the SMTP server should be down when an e-mail is to be sent, and there are no alternative SMTP accounts available, then the e-mail won't be delivered. If you want to add some robustness to Database Mail, and help ensure that the mail is delivered should the SMTP server go down for a short time, you can choose to increase this value to a higher number, such as 10. If you do this, and don't change any of the remaining settings, then Database Mail will try up to 10 times, waiting 60 seconds between tries, before it gives up.

You are now done, so click on **Next** to go to the summary screen, shown in Figure 2.10.

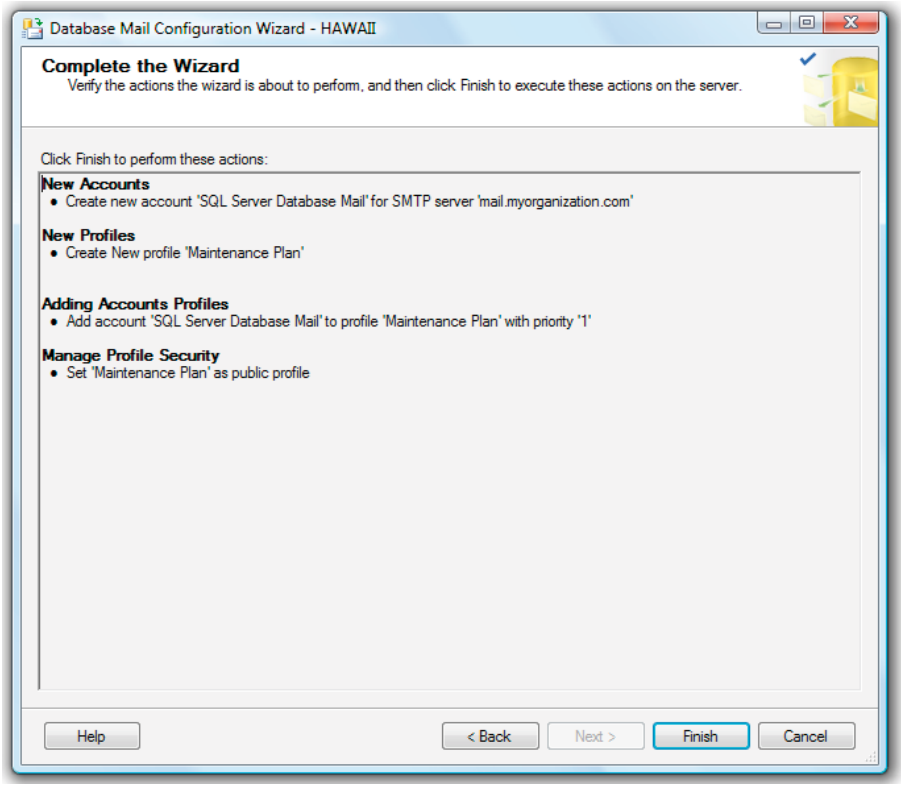

Figure 2.10: You are now done configuring Database Mail.

After a quick review of the summary screen, click on **Finish** and a final **Configuring...** screen appears, (Figure 2.11) indicating that your profile and its SMTP account have been created.

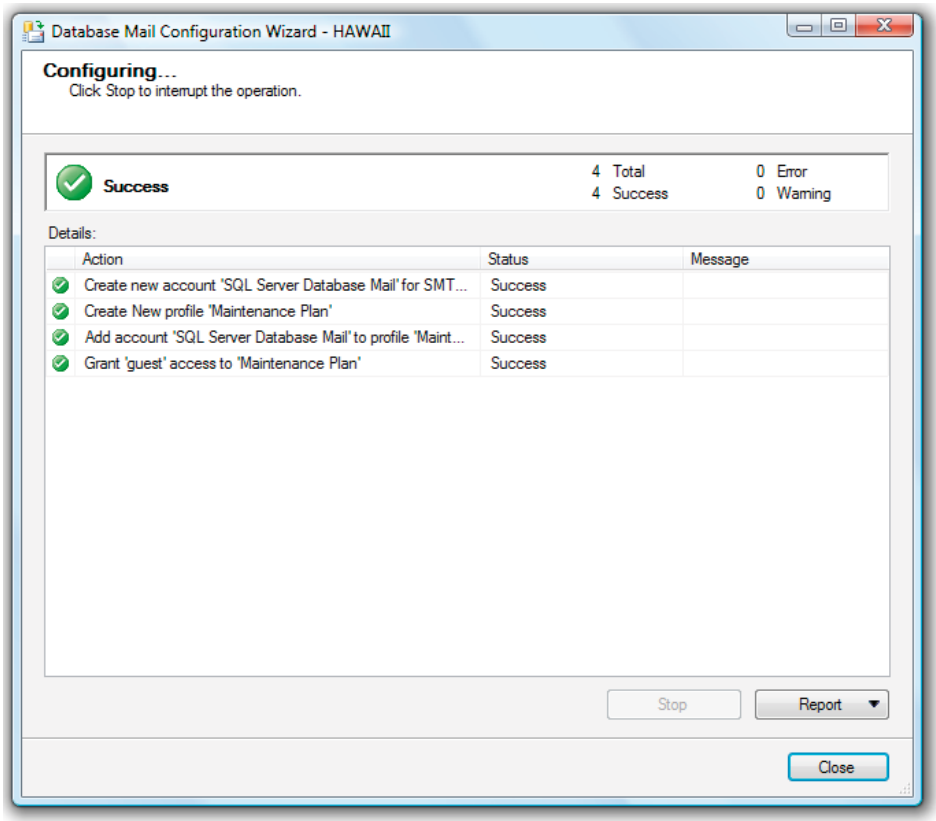

Figure 2.11: If all went well, you will see lots of "Success" entries.

If the **Configuring...** screen reports success, then Database Mail has been successfully set up for your SQL Server instance – or has it? While the success statuses are great, we still don't know if Database Mail has really been set up correctly. For example, perhaps there was a typo in the e-mail address or password, made when entering the SMTP information. If there was, Database Mail won't have any way of knowing this. In short, this means that you need to test your set up.

In order to test that Database Mail really is working as expected, close the **Configuring...** screen, then right-click on the **Database Mail** folder, just as you did when you began the Database Mail Wizard, and select **Send Test E-Mail**, as shown in Figure 2.12.

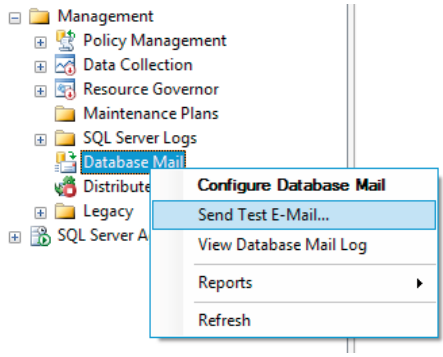

Figure 2.12: To ensure that Database Mail really works, you need to send a test e-mail.

The **Send Test E-Mail** screen, shown in Figure 2.13, will appear.

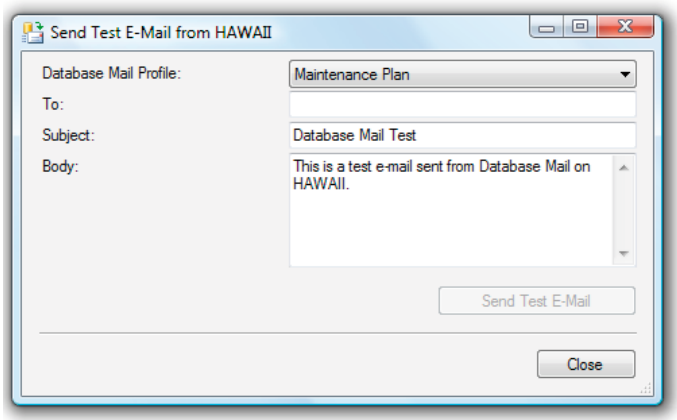

Figure 2.13: You need to enter your e-mail address to see if a test e-mail can be sent successfully from Database Mail.

Notice that the first option is **Database Mail Profile**, and it has a drop-down box next to it. This is used to select the profile you want use to send the test e-mail. In this case, you need to use the profile you just created, which was **Maintenance Plan**. If the profile you want to test is not selected, then you can choose it by selecting it from the drop-down box.

Fill in the **To** box with your e-mail address and click on **Send Test E-Mail** (this box will be grayed out until you enter an e-mail address). The screen shown in Figure 2.14 should appear.

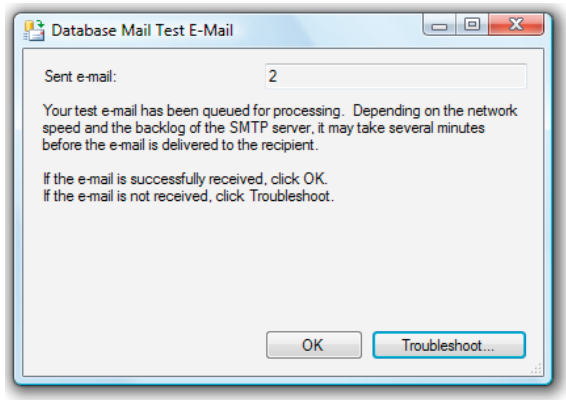

Figure 2.14: This screen can be used to help diagnose e-mail problems if you don't receive your test e-mail.

Having pressed **Send Test E-Mail**, the test e-mail will be sent to the designated account. The screen in Figure 2.14 tells you that it was sent (the number of the **Sent e-mail** is unimportant), so check your e-mail client and verify that the message was received. If the SMTP server is busy, or the e-mail client only polls for e-mails every few minutes, you may have to wait a short time before it appears. If you don't see the e-mail after a little while, be sure to check your spam folder to see if it ended up there.

Once you receive the test e-mail, you know Database Mail has been configured correctly and you are ready to continue with the next step, which is to set up an operator. If your mail never arrives, try clicking on the **Troubleshoot...** button, as shown in Figure 2.14, which sends you to articles in Books Online that guide you through the troubleshooting process.

How to Configure a SQL Server Agent Operator

When we configure a Maintenance Plan to send an e-mail, created with either the Maintenance Plan Wizard or the Maintenance Plan Designer, we aren't able to enter an e-mail address directly into the Maintenance Plan. Instead, we configure e-mails to be sent to an **operator**.

An operator is an alias for a specific person (such as yourself), or a group (such as a DBA mail group). This alias is more than just a name; it is actually a set of attributes that include the operator's name, the operator's contact information, and the operator's availability schedule.

Here's an example: let's say that a company has three DBAs, each working a different eight hour shift, so that all of the organization's SQL Servers have 24-hour DBA coverage. The DBAs are:

Name	Contact Information	Working Hours
Brad	brad@mycompany.com bradpager@mycompany.com	9 a.m. – 5 p.m.
Tony	tony@mycompany.com tonypager@mycompany.com	5 p.m. – 1 a.m.
Andrew	andrew@mycompany.com andrewpager@mycompany.com	1 a.m. – 9 a.m.

Each DBA can become an operator. For example, an operator could be created called "Brad" that includes his name, contact information, and his working hours. The same is true for the other DBAs. One advantage of using operators, instead of using specific e-mail addresses, is that if any piece of information changes about an operator, it can be changed in a single place.

If specific contact information was stored within Maintenance Plans, then every time some information changed, then all of the Maintenance Plans would have to be manually changed, which could be a lot of work.

In addition, since working hours can also be associated with an operator, it is possible to create a Maintenance Plan that is able to send e-mails to the right DBA, during their working hours. Of course, you don't need to take advantage of all this flexibility, but it is there if you need it.

Now that we know what an operator is, we need to learn how to create them, because a Maintenance Plan cannot be configured to use an operator until the operator has first been created and configured.

To create a new operator, open SSMS, navigate to the SQL Server instance you wish to configure, open up the **SQL Server Agent** folder, navigate to the **Operators** folder, right-click on it and select **New Operator**, as shown in Figure 2.15.

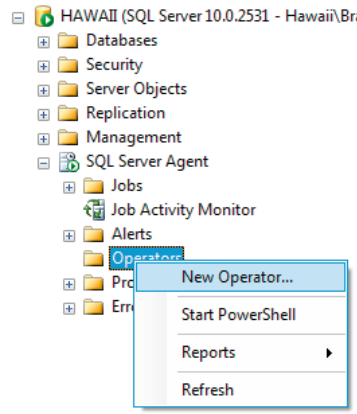

Figure 2.15: Operators are created using the SQL Server Agent.

The **New Operator** screen, shown in Figure 2.16 will appear.

While the **New Operator** screen has lots of options, we will focus only on the three that are most important.

- **Name** – this is your name, or the name of the person or group who you want to receive the Maintenance Plan e-mails.

- **Enabled** – this option is selected by default, and you don't want to change it, otherwise you won't be able to receive any notices from SQL Server.

- **E-mail name** – this option is poorly named. It really means that you are supposed to enter your e-mail address, or the group's e-mail address here.

That's it; all the other options are optional, and you can use them if you like, or leave them blank. When you are done, click on **OK**, and the name you specified in Figure 2.16 will now appear under the **Operators** folder in SSMS. If you have more than one person who should be notified of Maintenance Plan jobs, you can create additional operators. Alternatively, you could enter an e-mail group instead of an individual e-mail address, in the **E-mail name** field. This way, when a Maintenance Plan report is sent to a single operator, everybody in the e-mail group will receive the same e-mail.

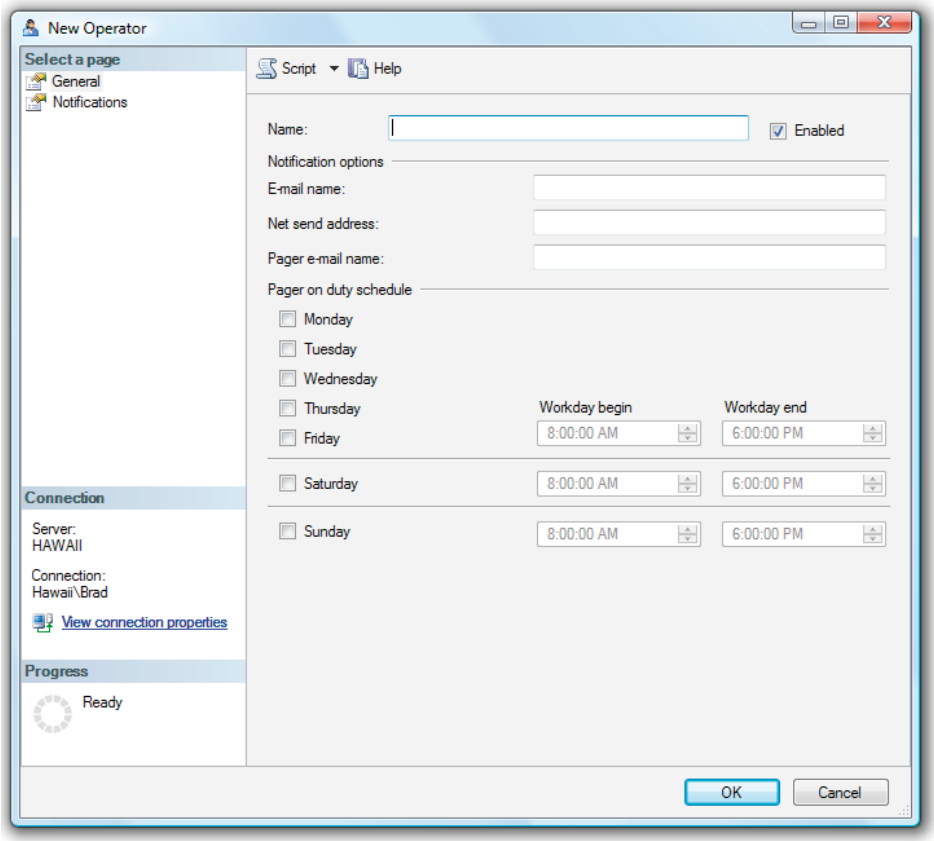

Figure 2.16: Most of the options on this screen are optional.

Summary

You now have set up Database Mail and at least one operator. Now, when you create a Maintenance Plan, you can choose to send e-mails to an operator. In the next chapter, I will show you how this is done.

Chapter 3: Getting Started with the Maintenance Plan Wizard

Now that we have all the preliminaries out of the way, we can focus on how to create a Maintenance Plan using the Maintenance Plan Wizard. The marketing story behind the Wizard is "answer a few simple questions on some screens, and voilà, you have a working Maintenance Plan." Of course, the reality is not quite that simple.

While the Wizard does present various configuration options, it does not explain what they do, or their pros and cons. There is also no way to determine if the options you choose are even applicable to your situation.

This chapter will provide a quick overview of the entire process of creating a Maintenance Plan from start to finish, using the Wizard. I'll cover preparation, how to start, and how to navigate through and complete the Wizard, using an example Maintenance Plan that implements the core database maintenance tasks identified in Chapter 1.

I won't be going into a lot of detail this time around. Instead, I will save the detail for later chapters, which will show you how to implement each individual task, from database integrity checking to reorganizing indexes and updating statistics, and cover every possible option available. As we progress through this and subsequent chapters, I'll try to anticipate and answer your likely questions, and offer lots of advice and best practices on database maintenance.

So, while this chapter will get you started, it is just a beginning. We still have a lot of hard work ahead before you know everything there is to know about using the Maintenance Plan Wizard.

Finally, before we start, I will remind you again here, as I will be reminding you throughout this book: a Maintenance Plan created with the Maintenance Plan Wizard or Maintenance Plan Designer provides only the core components of a SQL Server database maintenance plan. As discussed in Chapter 1, there are additional database maintenance tasks, which you must do outside of a Maintenance Plan created using the Wizard.

Exploiting the Full Potential of the Wizard

If you decide to use the Maintenance Plan Wizard to create Maintenance Plans for a given server or set of servers, my advice would be to take advantage of as many of its features as you can. The Wizard allows you to perform the core database maintenance tasks using one tool, and you might as well get all of the benefits the tools provides, such as the ease of setup.

I suggest that you avoid thinking that you'll use some of the features available in the Wizard, but perform other tasks using T-SQL or PowerShell scripts. Instead, pick one or the other, for a given server, or set of servers, and stick with it. For all the examples in this chapter and this book, I am going to assume that you will be taking maximum advantage of the features available to you from the Maintenance Plan Wizard.

Investigating Existing Maintenance Plans

Before you create a new Maintenance Plan, it is a good idea to first check to see if there are any existing Maintenance Plans, and find out what they do. This way, you can avoid inadvertently creating a new Maintenance Plan that includes tasks that overlap with an existing plan.

To find out if there are any existing Maintenance Plans, open SSMS, select the relevant SQL Server instance, and then click on the **Management** folder. The contents of the **Management** folder vary between SQL Server 2005 and SQL Server 2008. However, in either version, any current plans will be stored in the **Maintenance Plans** subfolder. If it's empty, it means there aren't any. Otherwise, you will see a list of one or more Maintenance Plans, as shown in Figure 3.1.

If there are any Maintenance Plans, you will want to check them out to see if they are performing as you expect. If they are, then you may not need to create new ones. On the other hand, if the existing Maintenance Plans are poorly designed, or don't perform all the tasks you want them to perform, then you may want to delete them and start from scratch. Often, starting over is easier to than trying to fix ill-conceived Maintenance Plans.

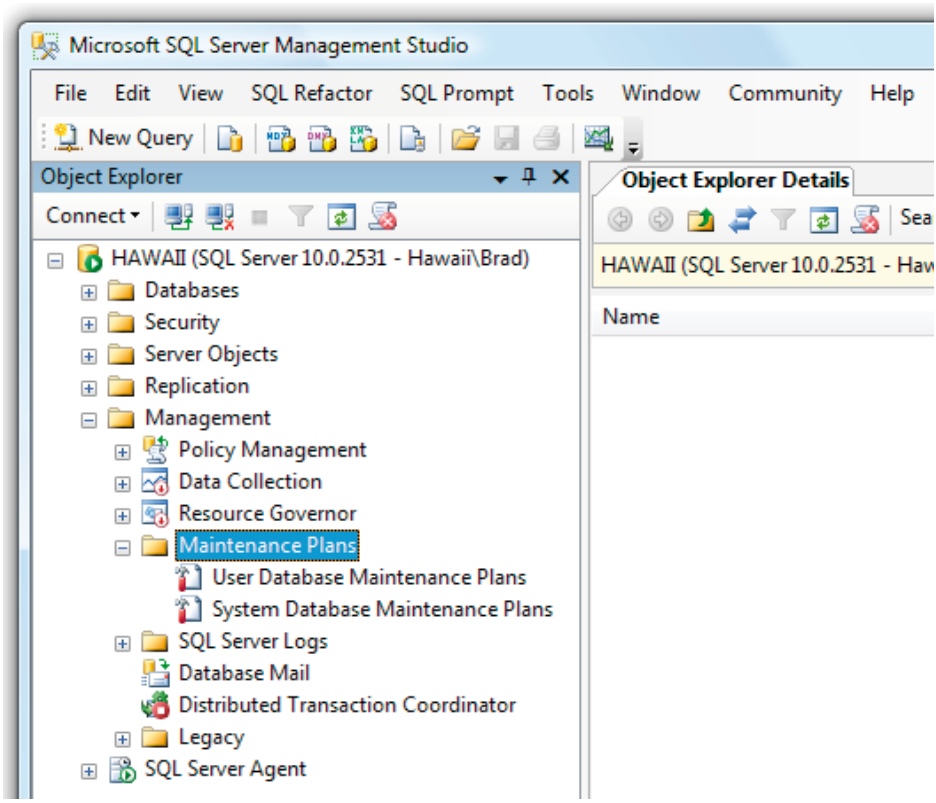

Figure 3.1: This example shows two existing Maintenance Plans.

Chapter 19 demonstrates how to view and modify the contents of an existing Maintenance Plan, using the Maintenance Plan Designer.

Privileges required for managing Maintenance Plans

In order to view Maintenance Plans from within SSMS, or to create or modify them, you have to be a member of the sysadmin fixed server role.

For now, let's assume that there aren't any existing Maintenance Plans and that you want to create a new one.

Creating a Maintenance Plan

In this section, I'll walk through all the Wizard steps required to create a Maintenance Plan that will perform all of the "core" database maintenance tasks identified in Chapter 1.

Starting the Maintenance Plan Wizard

When I use the Maintenance Plan Wizard to create a new Maintenance Plan, I often goof up and choose the wrong option. Hopefully, I can help you avoid this bad habit by teaching you how to do things correctly from the beginning.

To start the Maintenance Plan Wizard, open SSMS and navigate to the Maintenance Plans folder of the relevant server (as shown in Figure 3.1) and right-click on this folder to reveal the pop-up menu shown in Figure 3.2.

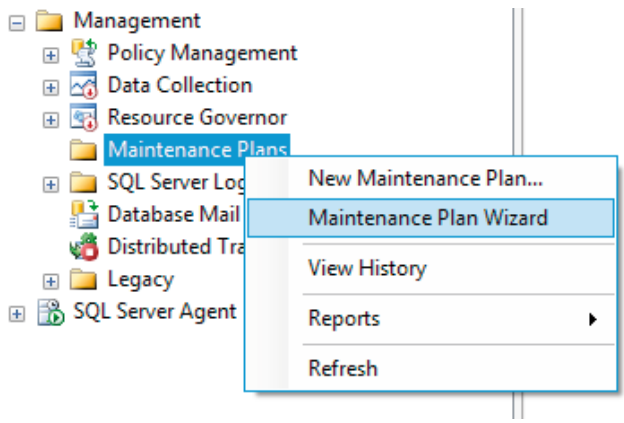

Figure 3.2: Choose Maintenance Plan Wizard from the menu.

The mistake I almost always make is to select the **New Maintenance Plan...** option. I think I do this because I instinctively know that my goal is to create a new Maintenance Plan, and the first option looks like it will work. Actually, it will work, but not as expected. The **New Maintenance Plan...** menu option will start the Maintenance Plan Designer. Instead, select **Maintenance Plan Wizard**, which is the way to start the Wizard.

Scheduling Maintenance Tasks

Once you've started the Wizard and are past the splash screen (assuming someone hasn't already helpfully ticked the **Do not show this starting screen again** box), you'll arrive at the **Select Plan Properties** screen, shown in Figure 3.3.

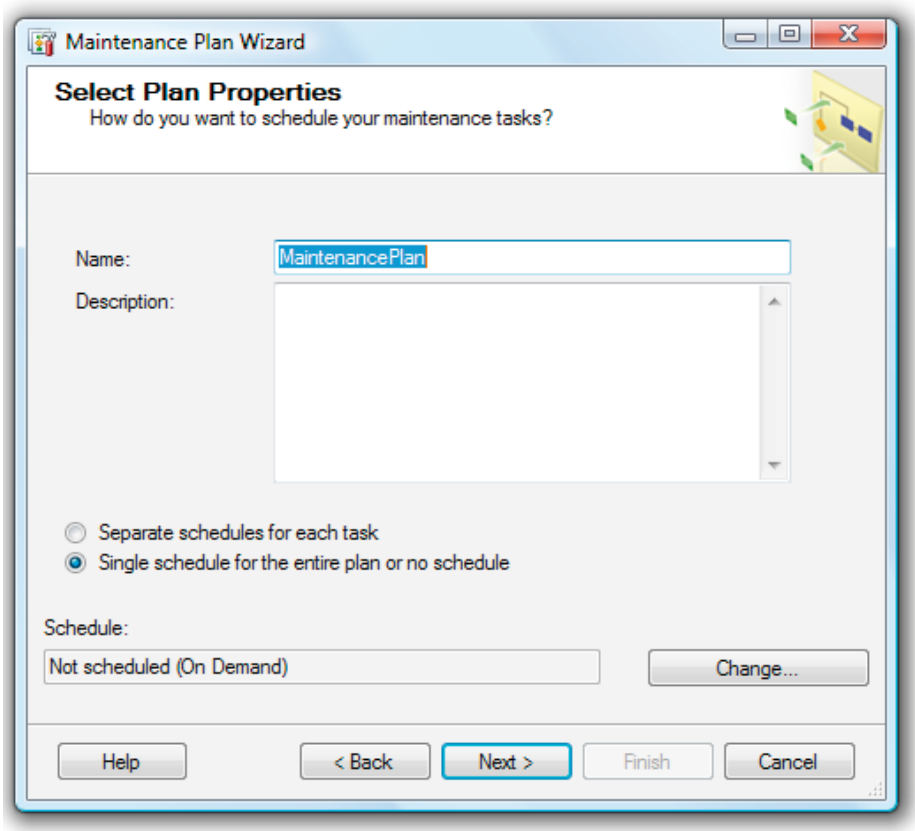

Figure 3.3: This screen is not as simple as you might think.

At first glance, the **Select Plan Properties** screen appears deceptively simple. In fact, it requires making an important choice that has significant implications later on. Let's examine this screen, one part at a time.

The first thing we need to do is to give our Maintenance Plan a name. The default name is "MaintenancePlan" but I strongly advise you to change this to something more descriptive of what the plan will do, and/or the databases it will affect. You and other DBAs must be able to distinguish one Maintenance Plan from another.

To change the name of an existing plan...

...Simply right-click on the plan in SSMS object explorer, and select the `Rename` *option.*

Next, we can add a description of your plan. This is optional, but it can be helpful to self-document the purpose of the plan.

Now, we arrive at the important choice I referred to just a moment ago. We need to choose between two options: **Separate schedules for each task** or **Single schedule for the entire plan or no schedule**. Notice that the second option is selected by default.

If you make the wrong decision now, and later change your mind about which option you want to use, you will have to delete your existing Maintenance Plan and recreate it from scratch (or you could correct the problem with the Maintenance Plan Designer, but it's a lot of work).

To help you determine which option you should select for your Maintenance Plan, I first need to explain what each option does.

Single Schedule or No Schedule

This is the default option and, if you choose it, all the tasks you create in your Maintenance Plan will be scheduled to run as a group, one after another, based on a single schedule of your choice. For example, if you schedule your Maintenance Plan to run once a week, then all the tasks you have selected for the Maintenance Plan will run only once a week. You can also select this option if you want to create a Maintenance Plan, but not schedule it, which allows you to manually run the Maintenance Plan instead.

If you are intending to do as many database maintenance tasks as possible in a single Maintenance Plan, the Single Schedule option is not a good choice because it will not allow you to perform different tasks at different times. For example, a task such as rebuilding indexes may only be run once a week, whereas full backups should be performed daily, and transaction log backups hourly.

While you could overcome this problem by creating multiple Maintenance Plans for each task, using a different schedule each time, this would, in effect, be the same as choosing the **Separate schedules for each task option**, only with a greater number of plans to manage.

In short, I recommend that you don't select this option. However, its one redeeming feature is that it insures that only one maintenance task runs at a time, so it prevents the potential problem of overlapping tasks, which could result in your SQL Server slowing down. This is harder to achieve using the **Separate schedules for each task** covered next, but it can certainly be done.

Separate Schedules for each Task

With this preferred option, the Wizard allows you to schedule each maintenance task independently. The inevitable downside, of course, is that it requires a little more work to determine an appropriate schedule for each task. For example, you will need to schedule when a task will execute, how often it executes, and very importantly, make sure you don't accidently schedule all the maintenance tasks to run at the same time, or to overlap each other. Scheduling is a manual process that is not always easy. However, it is perfectly feasible to prevent maintenance tasks from overlapping, when using this option, as I will demonstrate in the following chapters.

Based on the discussion up to this point, Figure 3.4 shows you the options I have selected.

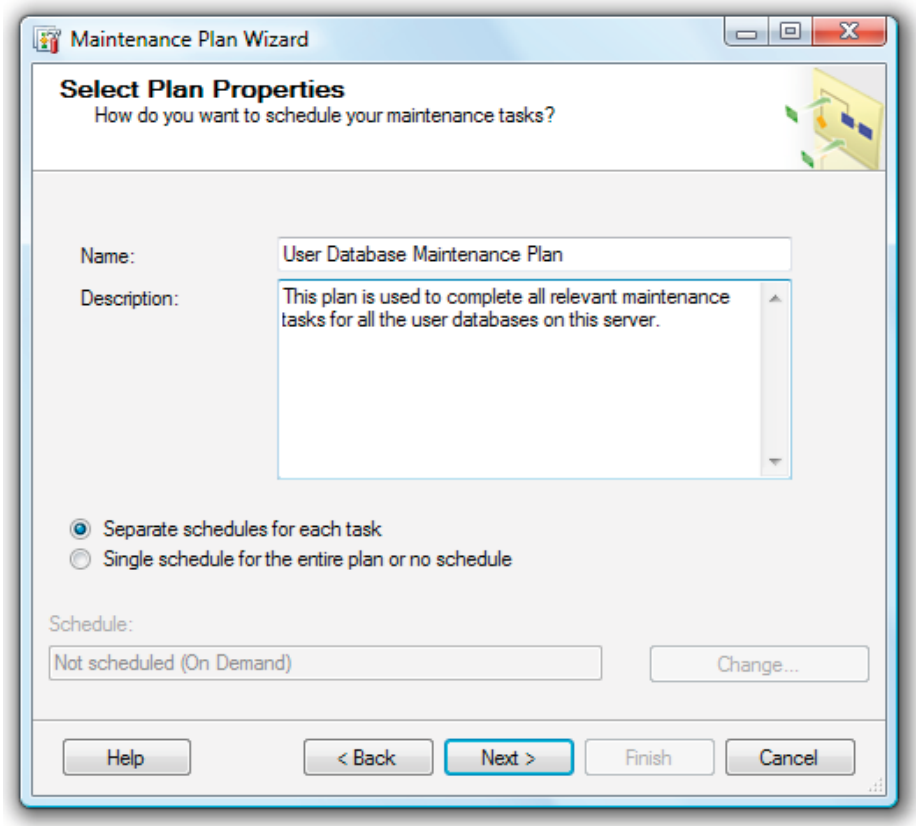

Figure 3.4: Once the Select Plan Properties page is complete, click on Next.

The reason I gave this plan the name **User Database Maintenance Plan** is because I generally prefer to perform maintenance on user and system databases separately. Although this is not

required, I choose to separate user and system database Maintenance Plans because the tasks I perform on them are somewhat different, and using two Maintenance Plans, instead of one, gives me more flexibility in the tasks I choose.

Notice that I also added a brief description, for documentation purposes, and selected **Separate schedules for each task**. In addition, note that the **Schedule** option is now grayed out because it is only applicable if **Single schedule for the entire plan or no schedule** is chosen.

Overview of Maintenance Tasks

Having established you plan properties, click **Next** and you'll arrive at the **Select Maintenance Tasks** screen, shown in Figure 3.5.

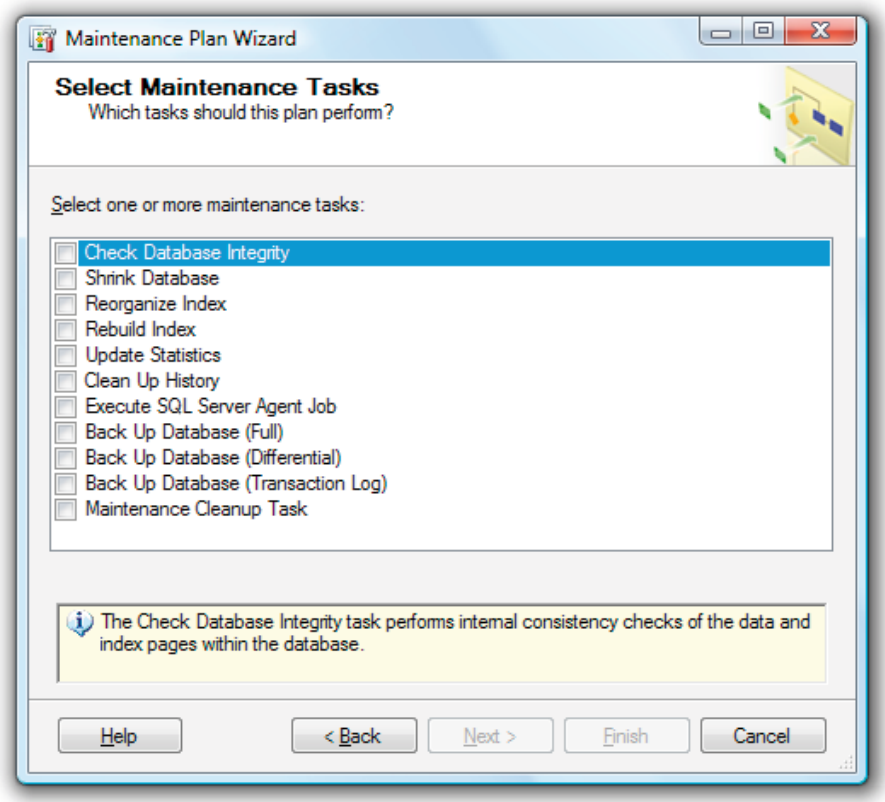

Figure 3.5: If you don't choose the correct maintenance tasks, you could create a Maintenance Plan that hurts your server's performance.

I know of novice DBAs who see the screen in Figure 3.5 and think to themselves, "I'm not sure what all these options mean, so I am going to choose them all, as more must be better than less."

Wrong. While the Maintenance Plan Wizard gives you the option of selecting all eleven of the maintenance tasks it offers, doing so can use server resources unnecessarily, and hurt its performance. We will learn a little of why this is true in this chapter, and then expand on the topic in subsequent chapters.

Up until this point I have been referring frequently to Maintenance Plan "tasks," but I have yet to explain what they really are, although I assume you have a general idea of what I have been talking about. Now, we get to find out what they are, and what they do. A database maintenance task is simply a specific activity performed as part of a Maintenance Plan. I will offer a brief description of these tasks now, so you get an idea of what each task does, but I'll save the details for upcoming chapters. In fact, following this chapter, are dedicated chapters on each of the following eleven maintenance tasks.

Check Database Integrity

The `Check Database Integrity` task runs `DBCC CHECKDB` against selected databases and performs an internal consistency check on them to see if there are any problems with their integrity. While this task is very resource intensive, it is critical that you perform it on a regular basis, to ensure that your databases aren't damaged.

Shrink Database

Never use the `Shrink Database` task. Is that clear enough advice? While we will discuss why it is not a good idea to automatically shrink a database in Chapter 6, the point to keep in mind is that, if you ever need to shrink a database, it should be done manually.

Rebuild Index

The `Rebuild Index` task runs the `ALTER INDEX` statement with the `REBUILD` option on indexes in the selected databases, by physically rebuilding indexes from scratch. This removes index fragmentation and updates statistics at the same time. If you use this option, you do not want to run the `Reorganize Index` or the `Update Statistics` task, as doing so would be redundant.

Reorganize Index

The `Reorganize Index` task runs the `ALTER INDEX` statement with the `REORGANIZE` option on the indexes in the selected databases. This task helps to remove index fragmentation, but does not update index and column statistics. If you use this option to remove index fragmentation, then you will also need to run the `Update Statistics` task as part of the same Maintenance Plan. In addition, you won't need to run the `Rebuild Index` task, as the use of `Reorganize Index` task (followed by the `Update Statistics` task) renders redundant the `Rebuild Index` task.

Update Statistics

The `Update Statistics` task runs the `sp_updatestats` system stored procedure against the tables of the selected databases, updating index and column statistics. It is normally run after the `Reorganize Index` task is run. Don't run it after running the `Rebuild Index` task, as the `Rebuild Index` task performs this same task automatically.

Execute SQL Server Agent Job

The `Execute SQL Server Agent Job` task allows you to select SQL Server Agent jobs (ones you have previously created), and to execute them as part of a Maintenance Plan. This feature offers you additional flexibility when performing database maintenance using the Maintenance Plan Wizard.

History Cleanup

The `History Cleanup` task deletes historical data from the `msdb` database, including historical data regarding backup and restore, SQL Server Agent and Maintenance Plans. If you don't perform this task periodically then, over time, the `msdb` database can grow very large.

Back Up Database (Full)

The `Back Up Database (Full)` task executes the `BACKUP DATABASE` statement and creates a full backup of the database. You will probably want to run this task daily against your system and production databases. In most cases, the databases you will be backing up with this task use the Full Recovery model, and you will also want to run the `Backup Database (Transaction Log)` task as part of your Maintenance Plan.

Back Up Database (Differential)

The `Back Up Database (Differential)` task executes the `BACKUP DATABASE` statement using the `DIFFERENTIAL` option. This task should only be used if you need to create differential backups.

Backup Database (Transaction Log)

The `Backup Database (Transaction Log)` task executes the `BACKUP LOG` statement, and, in most cases, should be part of any Maintenance Plan that uses the `Back Up Database (Full)` task. It is a common practice to run this task every hour or so, depending upon your needs.

Maintenance Cleanup Task

The `Maintenance Cleanup` task is problematic as it does not really do what it is supposed to do. In theory, it is designed to delete older backup files (`BAK` and `TRN`), along with older Maintenance Plan text file reports (`TXT`) files that you no longer need. The problem is that it can only delete one type of file at a time within a single Maintenance Plan. For example, if you choose to delete older `BAK` files, it won't delete older `TRN` or `TXT` files; if you choose to delete older `TRN` files, it won't delete older `BAK` or `TXT` files.

What we really need is a task that performs all three inside the same Maintenance Plan, but we don't have it. So, what is the best way to delete old `BAK`, `TRN`, and `TXT` files? One way is to use the Maintenance Plan Designer, which allows you to create three separate subplans that will take care of deleting each of these three kinds of files within a single Maintenance Plan (see Chapter 17). However, if you want to use the Maintenance Plan Wizard exclusively to delete all three file types, you must create three different plans to accomplish your goal.

Selecting Core Maintenance Tasks

Now that we know a little bit about each of the eleven maintenance tasks available to us from within the Maintenance Plan Wizard, let's take a closer look at the **Select Maintenance Tasks** screen, which allows you to select the specific maintenance tasks that you'd like to include as part of the plan. Which options you choose will depend on your goals for the Maintenance Plan, along with any special needs of your databases. To keep the example simple, I am going to only select the tasks illustrated below in Figure 3.6. Don't worry if I have left out one of your favorites, as I will discuss each task in later chapters. The goal, for now, is to provide only a high-level overview of the Maintenance Plan Wizard.

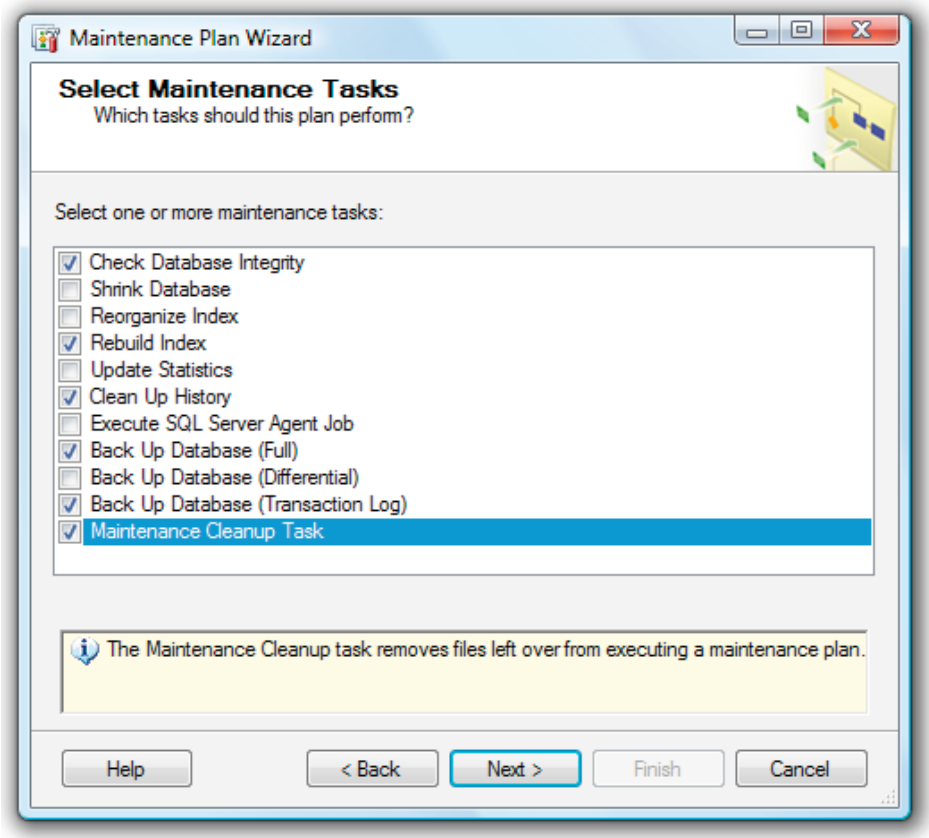

Figure 3.6: The above tasks are commonly selected when creating a Maintenance Plan using the Maintenance Plan Wizard.

Note that, despite the previous discussion regarding its limitations, I chose to include the **Maintenance Cleanup** task, for illustrative purposes.

Maintenance Task Order

Having chosen the maintenance tasks you want to include in your Maintenance Plan, click **Next**, and the **Select Maintenance Task Order** screen appears, as shown in Figure 3.7.

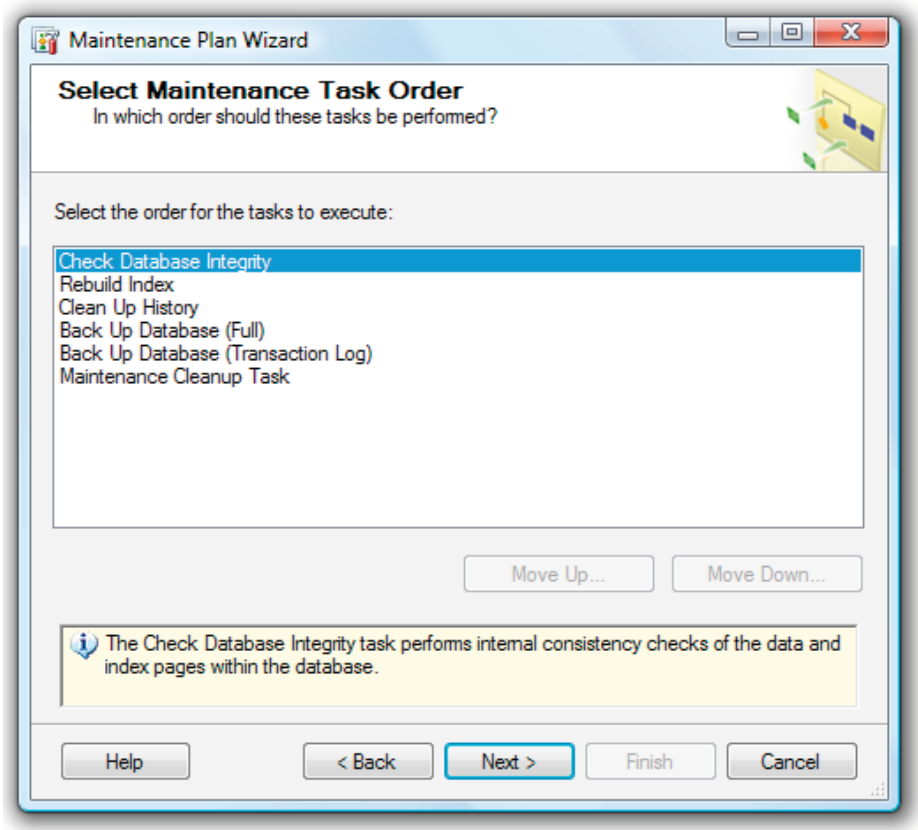

Figure 3.7: You must tell the Maintenance Plan Wizard the order in which you want it to perform its tasks.

In this screen, you must tell the Wizard the order in which you want the maintenance tasks to execute. Figure 3.7 shows the six tasks we previously selected, in default order. In many cases, the default order will be fine, but you can move them up or down by selecting any one of the tasks, one at a time, and then clicking on either the "Move Up" or the "Move Down" button.

This option is really only useful if you choose the **Single schedule for the entire plan or no schedule** option instead of the **Separate schedules for each task** that I recommend you use. Why? If you choose **Single schedule for the entire plan or no schedule**, this option is important because there is only one schedule, and each task within a Maintenance Plan will run one after another (based on your ordering), until the plan is complete. If you choose **Separate schedules for each task**, the order of the tasks is dependent on the order you schedule them, overriding any order you specify in this screen. This is because each task has its own schedule. More on this in a moment.

Whichever scheduling option you choose, consider the following when selecting the order of execution of the maintenance tasks.

- **Logical Task Ordering**. A task such as **Clean Up History** can be performed at any point in the plan but, for other tasks, there is a certain logical order in which they should be performed.
    - It makes sense to start the Maintenance Plan with the **Check Database Integrity** task, because there is no point in running the rest of the maintenance tasks if the integrity of your database is in question.
    - The **Back Up Database (Full)** should come before the **Backup Database (Transaction Log)** task as we can't perform a transaction log backup before we perform a full database backup. If we were to try, we would get an error.
    - If a **Rebuild Index** task (or the **Reorganize Index** and **Update Statistics** tasks) is performed during the same maintenance window as the **Back Up Database (Full)** task, then I always perform the **Rebuild Index** task first. Why? This way, should I need to perform a restore of the backup, it will have its indexes defragmented and will be ready for production.
    - The **Maintenance Cleanup** task, if selected (see previous discussion), should be performed only after the **Back Up Database (Full)** has been completed. This way, you can ensure that a good backup has been made before deleting any older backups.
- **Task Scheduling**. If you choose **Separate schedules for each task**, the scheduling of these tasks (covered later) will determine the *actual* order in which they occur. For example, it is possible to schedule the tasks in such a way that the `Backup Database (Transaction Log)` task runs before the first ever `Back Up Database (Full)` task for a given database, although this would cause the Maintenance Plan to fail. The key thing to remember is that the logical task order you select in Figure 3.7 is not absolute, and that it can be overridden by the schedules you set for each one.

In this example, we will accept the default order, and click **Next**.

Configuring Individual Tasks

At this point, if we were continue with this example, we would be presented with six **Define...Task** screens, one after the other, allowing you to configure the exact characteristics of each task. The first screen presented in this example, will be the **Define Database Check Integrity Task**screen, shown in Figure 3.8.

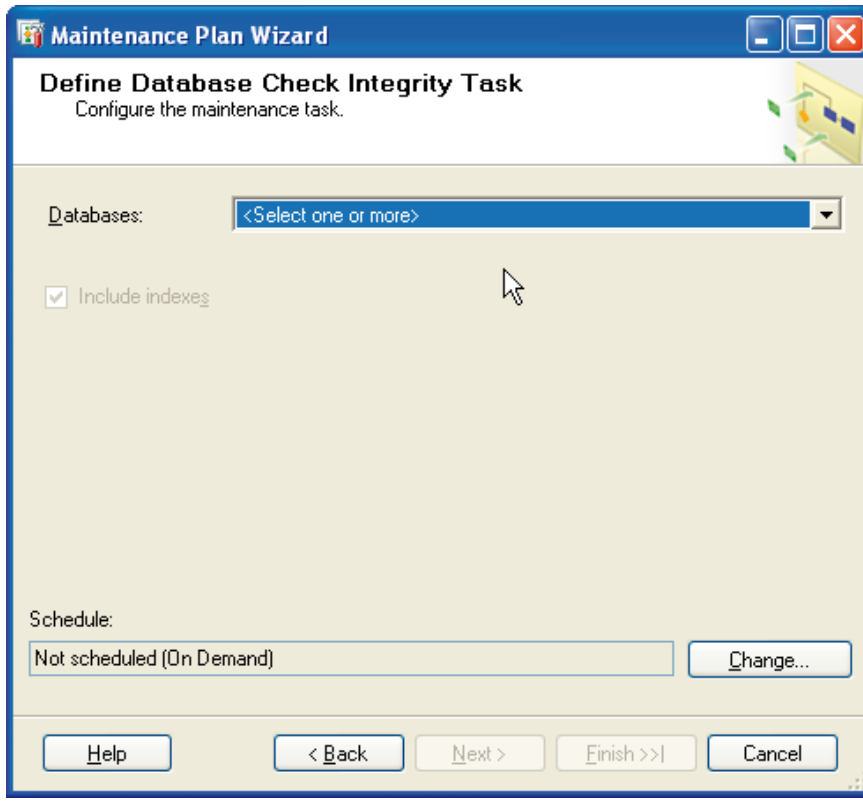

Figure 3.8: The Define Database Check Integrity task configuration screen.

The basic layout of this screen is more or less the same for every task.

- **Database Selection** – a drop-down box to choose the database(s) on which the task will execute. This step is common to most tasks.
- **Task-specific configuration options** – a set of configuration options specific to a given task.
- **Task Scheduling** – an option to schedule the task. This step is common to every task, assuming that you chose **Separate schedules for each task** when you first started the Wizard.

Although this screen looks simple, it is hiding a lot of detail. Let's take a look at each of the three sections, although we'll save a lot of these details for later chapters.

Database Selection

As you might expect, the first step when configuring most tasks is to specify the database or databases on which the maintenance task to act. The **Databases** drop-down box appears on the configuration screen for all tasks that can be configured through the Wizard, with the exception of the `Execute SQL Server Agent Job` (Chapter 10), `History CleanUp` (Chapter 11) and `Maintenance Cleanup` (Chapter 15) tasks, where database selection is not necessary. In other cases, the database selection process for a task (for example the `Rebuild Index` task, see Chapter 7) is slightly more complex, as you will be offered the chance to narrow the scope of the task to specific objects within a selected database. However, most often you will want a task to act on the database as a whole.

For all tasks where database selection is relevant, most of the other options on the screen will be grayed out until you select at least one database.

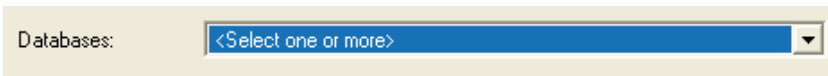

Figure 3.9: The database selection drop-down box.

So let's take a close look at how you use this option, as a lot of it is hidden away from us at the moment. To select which databases you want to run this task against, click on the **<Select one or more>** drop-down box, and the screen shown in Figure 3.10 appears. This screen allows us to make one selection from four different choices.

All databases

This option means exactly what it says. If you select this option, every database on your SQL Server instance will have the task run against it. One advantage of using this option is that it covers both user and system databases and, if you add any databases at any time, even after the Maintenance Plan has been created and is in production, they will automatically be included when this task runs. As such, this option allows you to create a single Maintenance Plan to cover all of your databases.

On the other hand, if you include all your databases in a single Maintenance Plan, you have less flexibility. This means that, although selecting this option may seem to be the best choice, it may not actually be the best, because you may need to treat some databases differently from others. We will be talking more about this later in the book.

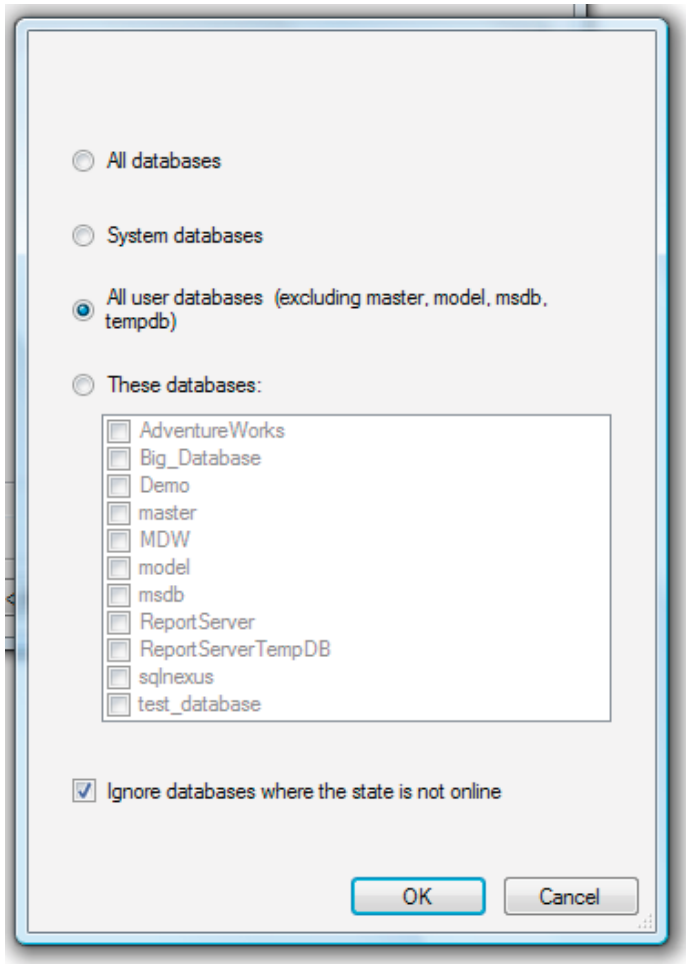

Figure 3.10: The Maintenance Plan Wizard needs to know which databases you want to run the task against.

System databases

This option specifies that your task will only run against the SQL Server instance's system databases, which include `master`, `model`, and `msdb` (but not `tempdb`).

This option is commonly used to create a specific Maintenance Plan for your system databases. Generally, the maintenance tasks you want to perform on system databases are a little different than those you run against production user databases. For example, system databases use the Simple Recovery model, and so transaction log backups can't be carried

out. On the other hand, most production databases use the Full Recovery model, which allows transaction log backups to be made. This and other differences between system and production database maintenance, mean it is common to create separate Maintenance Plans for each class of databases.

> ***tempdb and Maintenance Plans***
>
> `tempdb` *is not included in a Maintenance Plan because it is automatically deleted and recreated every time SQL Server is restarted.*

All user databases...

Just as a DBA often wants to create a Maintenance Plan that is specific to system databases, so he or she will often create a plan dedicated to the maintenance of all production user databases. A nice feature of this option is that it will automatically include all user databases, even ones that you create after the Maintenance Plan is created. This is handy because you don't have to worry about modifying the Maintenance Plan if you subsequently add or delete databases from your SQL Server. This is the option that we'll use for most of the examples in this book, as shown in Figure 3.10.

These databases

This final option allows you to build a custom list of databases, both user and system, to which your task will apply. This option is often used when you want to treat a particular database differently from the rest of the databases. For example, you may have a policy that, for databases under a given size, the `Rebuild Index` task is used, but that the `Reorganize Index` and the `Update Statistics` tasks are preferred on any databases above that size. This distinction is sometimes made if the `Rebuild Index` task on a large database takes longer to execute than the available maintenance window, which could end up blocking users trying to access the database. As we will learn later in this book, the `Reorganize Index` and the `Update Statistics` tasks, combined, perform a similar function to the `Rebuild Index` task, but do so without blocking users.

A disadvantage of this option is that the databases you select here aren't dynamic. By this, I mean that databases that are created or deleted subsequent to the creation of the plan are not automatically added to, or removed from, that plan. In the latter case, this will mean that an error will occur when the Maintenance Plan is run. You have to manually edit the Maintenance Plan every time you want to add or delete a database to be used by the plan.

Ignore databases where the state is not online

Finally, on this screen is an **Ignore databases where the state is not online** checkbox which, by default, is activated. This means that any databases that are offline at the time when the plan is running will be ignored. If you uncheck this box, and a maintenance task attempts to run against an offline database, then the Maintenance Plan will fail when it is executed because it will not be able to complete its operation.

Generally speaking, I suggest you leave this option activated to ensure that the plan works successfully on all online databases.

> ***Ignore databases where the state is not online…***
>
> *…is only available in SQL Server 2008. If you run a SQL Server 2005 Maintenance Plan, and it includes tasks to be run against an offline database, then the Maintenance Plan will fail.*

Task-Specific Configuration Options

Click on **OK** to be returned to the **Define Database Check Integrity Task** screen, with the database choice displayed in the drop-down box, as shown in Figure 3.11.

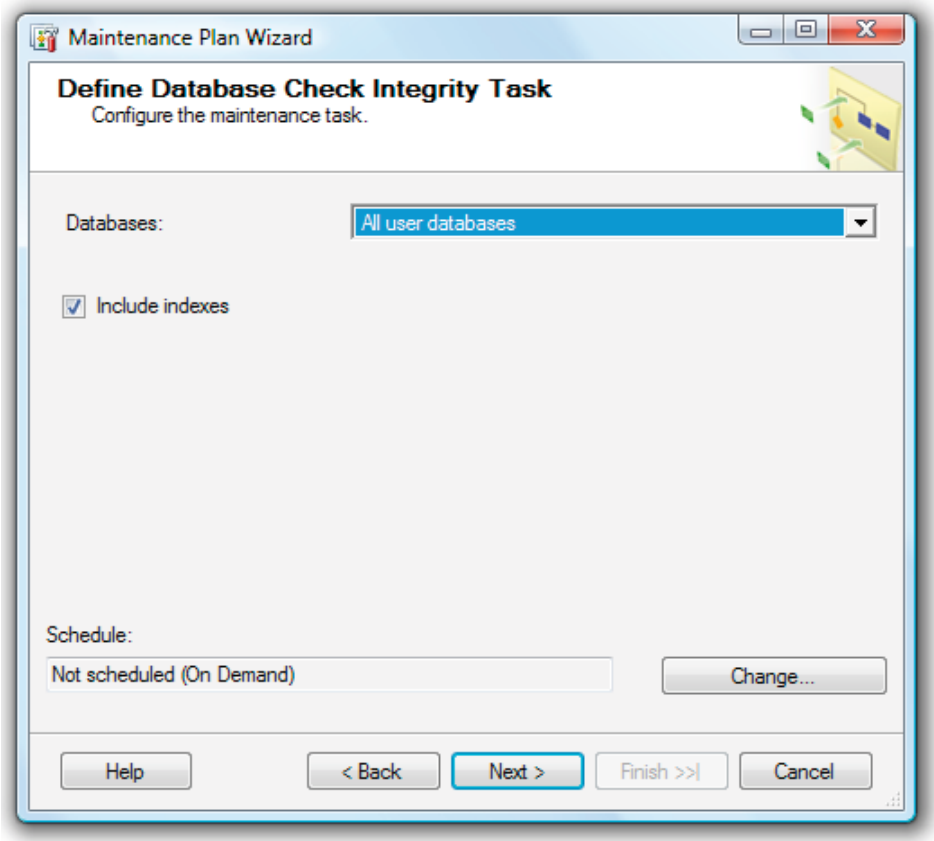

Figure 3.11: Once at least one database is selected, additional options are the screen can be configured.

The middle section of the screen in Figure 3.11 contains the configuration options for a given task. In this example, there is only one option, **Include indexes**. Each task will have different options. For example, Figure 3.12 shows the same portion of the screen for the `Rebuild Index` task.

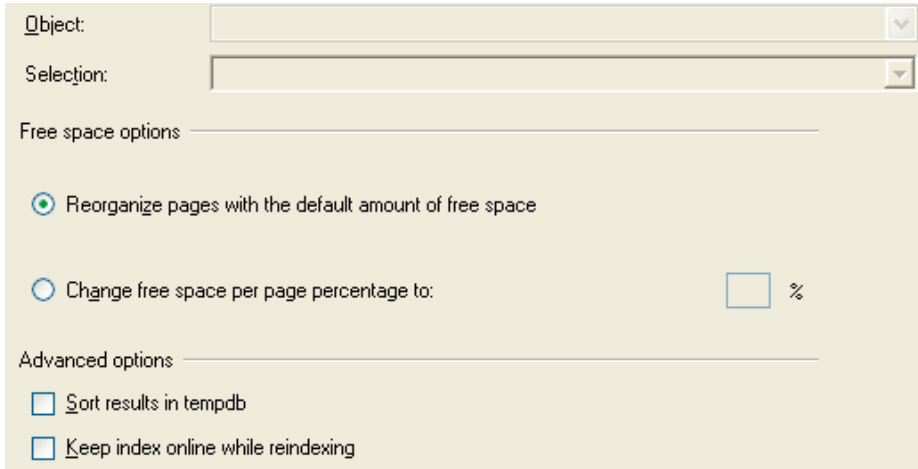

Figure 3.12: Task-specific configuration options for the Rebuild Index task.

If I were to go through all the options on this screen in full detail, and the equivalent screens for the remaining tasks, it would quickly become overwhelming. Instead, starting with Chapter 5, I have devoted individual chapters to each of the eleven possible database maintenance tasks. Please refer to a task's specific chapter for full details on all of these configuration options.

Task Scheduling

The final section of the screen shown in Figure 3.11 is devoted to task scheduling, as shown in Figure 3.13.

Figure 3.13: Scheduling maintenance tasks.

By default, each task is set to **Not Scheduled (On Demand)**. To implement a schedule on which the task should run, click on **Change** and you'll arrive at the **Job Schedule Properties** screen, shown in Figure 3.14.

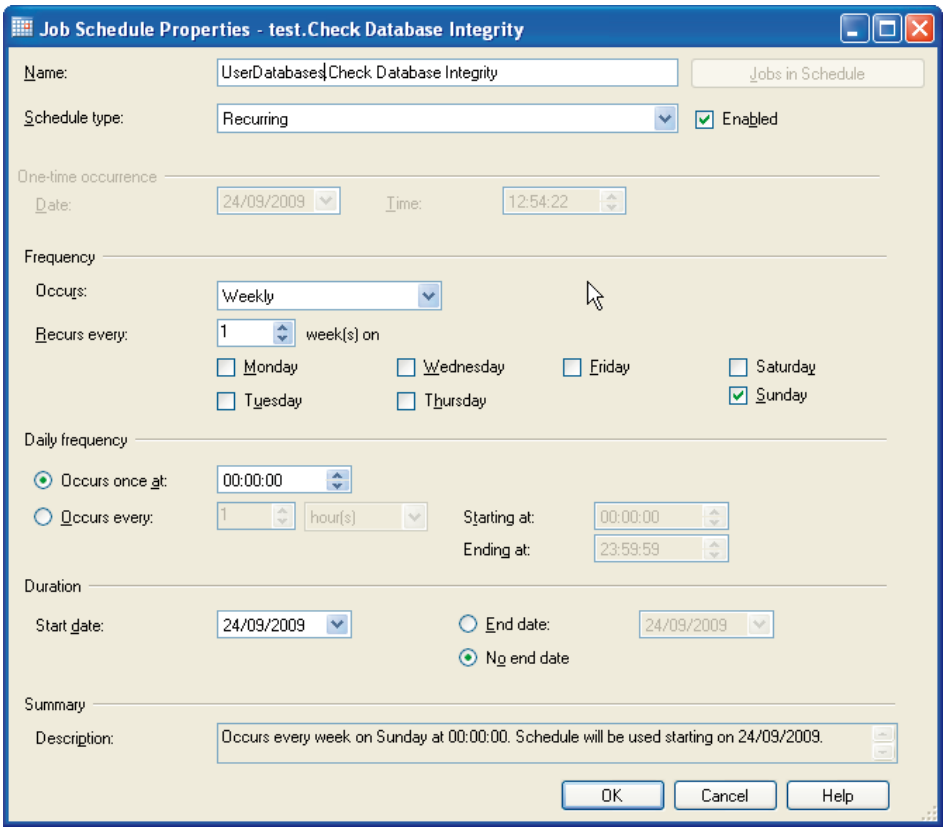

Figure 3.14: The Job Schedule Properties screen.

As you can see, this is a complex screen with many options, and some of these options do not mean quite what you might initially think. As a result, we will defer a full discussion of all of these options to a dedicated chapter (Chapter 4), where we'll also look at general considerations when scheduling multiple tasks, so as to avoid overlap and server resource contention.

For now, you can either click **Cancel** to exit the **Job Schedule Properties** screen, and then exit the Wizard; or you can work through the screens, leaving all tasks unscheduled, selecting databases where necessary, and accepting all the default task configuration options, until you arrive at the **Select Report Options** screen.

Report Options

Once you've configured and scheduled each of the individual tasks that make up a Maintenance Plan, you will arrive at the **Select Report Options** screen. After a Maintenance Plan runs, it can produce a report of the tasks that were performed. For example, you can see the results of a `Database Check Integrity` task, or see what databases were backed up. These reports are very useful because they tell you whether or not everything ran correctly, and they are helpful when troubleshooting Maintenance Plans that don't seem to be doing what you expect them to do.

Chapter 2 described how to configure Database Mail so that these Maintenance Plan reports can be sent to you via e-mail, and it is when we reach the **Select Report Options** screen, shown in Figure 3.15, that our groundwork pays off.

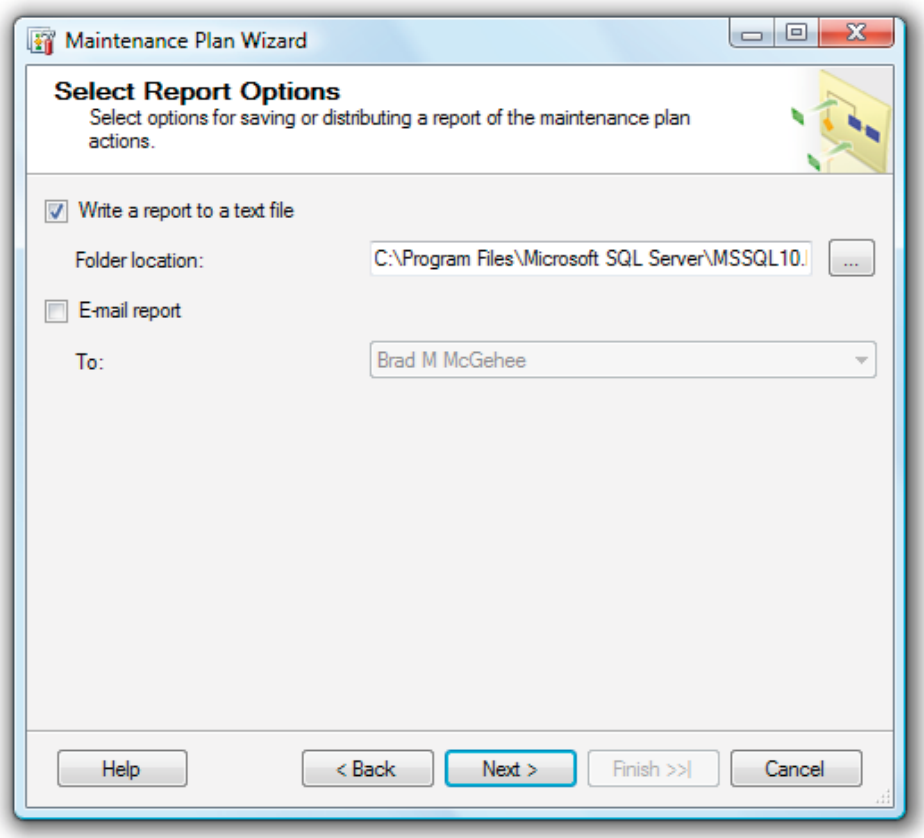

Figure 3.15: After a Maintenance Plan runs, it can write a report to the local SQL Server, and/or send you an e-mail message containing the report.

By default, the **Write a report to a text file** option is selected, and I recommend you keep it selected. This way, every time a Maintenance Plan runs, a new report will be produced that you can view to see what is going on. By default, in SQL Server 2008, Maintenance Plan reports are written to the `\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Log` folder. Unless you have a good reason to place the reports in another location, I suggest you keep them here, as this is the standard practice. If you place them elsewhere, it will make them harder for other DBAs to find.

Removing/Archiving Maintenance Plan reports...

Unless you create a Maintenance Task to remove them, all plans will remain in the designated folder indefinitely. If you forget to delete them, you can end up with a folder with thousands of old reports. You can use the Maintenance Cleanup task (see Chapter 15 for details) to periodically remove older files.

The **E-mail report** option on the screen is not selected by default. As we discussed in Chapter 2, in addition to the text report saved on disk, I think it is a good idea to have these reports sent to you via e-mail, so that you can more easily review them. Not only does this save you the effort of locating the correct report among all of your servers, the e-mails also serve as a constant reminder that you need to read them to see if everything is running as it should be.

Note

Even if you have reports e-mailed to you, I suggest you have the reports written to disk as well. That way, should you need to troubleshoot a problem, all of the reports will be in a single location.

Notice that my name is listed in the drop-down box next to the **To** option, reflecting the fact that I set myself up as an operator. If you have not set up Database Mail or created any operators, then no names will be available from the drop-down box.

To receive e-mail reports every time this Maintenance Plan runs, select the **E-mail report** options and select your name from the drop-down box, as shown in Figure 3.16.

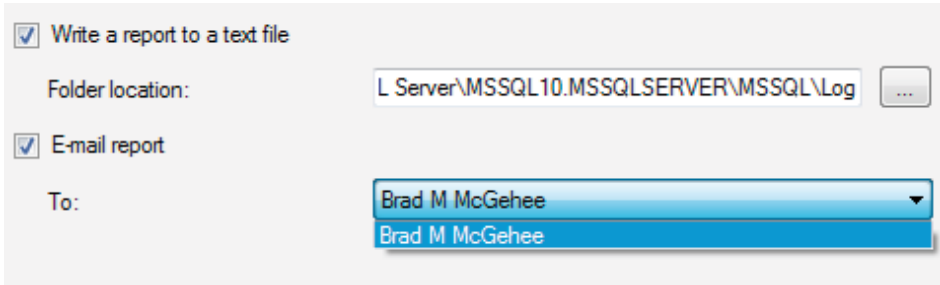

Figure 3.16: Since I am the only Operator, there are no other names to choose from.

As you will see, you can only select a single operator. If you want to send Maintenance Plan reports to multiple users, you will need to have set up an operator that uses a group e-mail account, in which case each e-mail will be sent to every member of the group, each time the plan runs. Note also that, just because you receive an e-mail report after a Maintenance Plan executed, you shouldn't necessarily assume that the plan executed *successfully*. You'll need to delve into the details of the report to ensure that each task within the plan did, indeed, complete as expected.

In Chapter 17, covering the Maintenance Plan Designer, we will discuss an alternative e-mail notification option that makes it easier to notify operators of problems with Maintenance Plans.

Completing the Wizard

After completing the **Select Report Options** screen, click **Next**, and the **Complete the Wizard** screen, shown in Figure 3.17, displays all the tasks you have configured, and also allows you to drill down to see what options you have set. I don't find this screen very useful, as I already know what I just did.

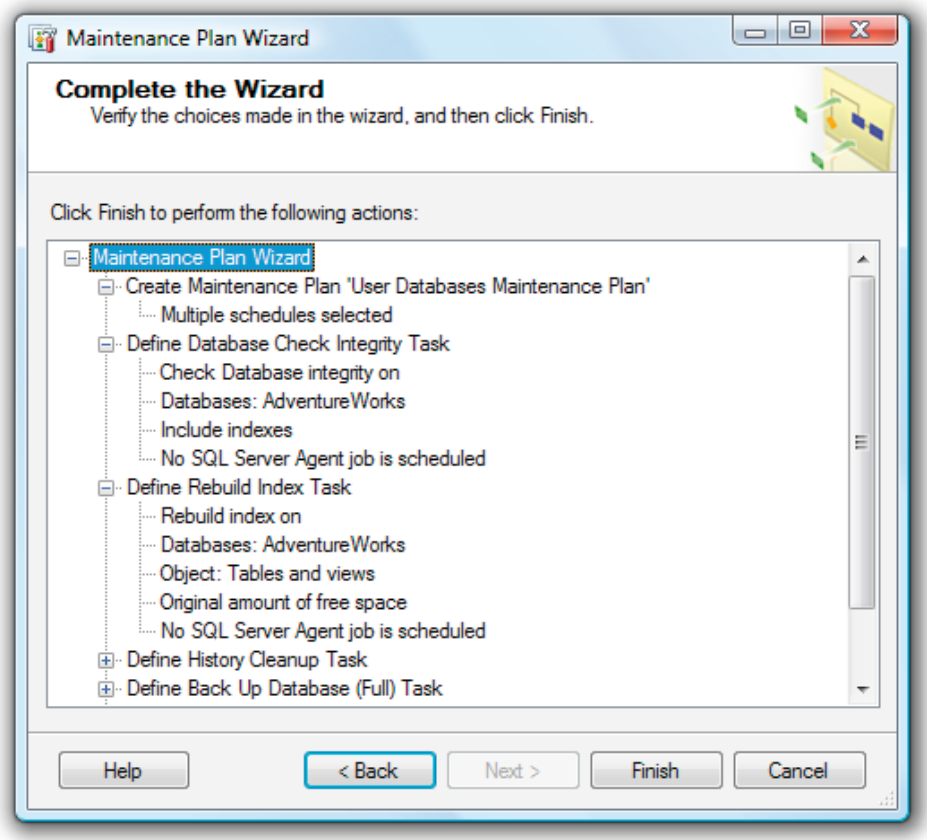

Figure 3.17: You can drill down into each of the tasks on this screen to see what settings you have configured.

At this point, click on **Finish** to create the Maintenance Plan and reach the very last Maintenance Wizard screen, as shown in Figure 3.18.

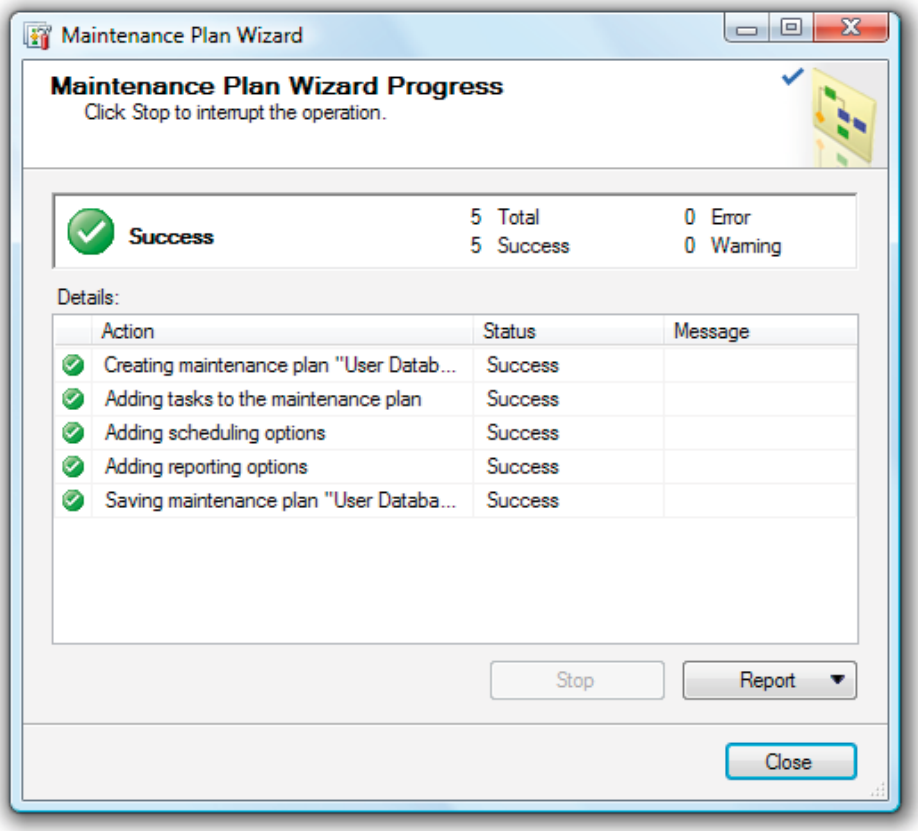

Figure 3.18: Hopefully, you will see all successes.

The Wizard creates the Maintenance Plan, and the **Maintenance Plan Wizard Progress** screen tells you if all the steps were successful. If you see all successes, you are done, and can now test your Maintenance Plan.

If you encounter any errors or warnings, a message link will appear next to the problem. Given the large number of potential warnings or errors you could get, it is not possible to cover them here. However, in most cases, the message link will provide you with a clue as to what the problem is, and you will have to figure out how to go about fixing it.

Having viewed this final screen, click on **Close** and we are done.

A Closer Look at Maintenance Plan Implementation

So, when you create a Maintenance Plan, what happens from an architectural point of view within SQL Server? In other words, how is the plan physically implemented? Each Maintenance Plan is implemented as a single SQL Server Integration Services (SSIS) package. There will be one Maintenance Plan SSIS package for every Maintenance Plan you create. This package is executed using one or more scheduled SQL Server Agent jobs that are automatically created.

We can view our new **User Databases Maintenance Plan** SSIS package by navigating to the relevant Maintenance Plans directory. To do this, open up SSIS from SSMS, navigate to the **Stored Packages** folder, then open up the **MSDB** folder, and finally, open up the **Maintenance Plans** folder, as shown in Figure 3.19.

Connecting to SSIS

To be able to view the SSIS packages, the SSIS service must be installed and running on your server, and then manually connect to the SSIS service from within SSMS.

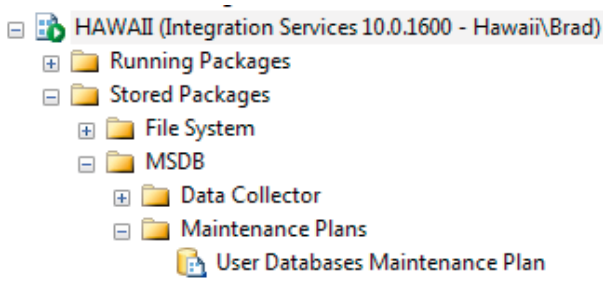

Figure 3.19: Viewing the new Maintenance Plan from within SSMS.

While you can view the Maintenance Plan SSIS packages from here, you can't view their contents or modify them here. To do this, you must open them using the Maintenance Plan Designer, which we will cover in Chapters 16 to 19.

While only one SSIS package is created per Maintenance Plan, one or more SQL Server Agent jobs will be created to run the package. If you selected **Single schedule for the entire plan or no schedule** then there will only be one SQL Server Agent job. However if, as advised,

you selected **Separate schedules for each task**, there will be a separate SQL Server Agent job created for each of the scheduled tasks in your Maintenance Plan.

To view the jobs created when you create a new Maintenance Plan, use SSMS to open up **SQL Server Agent** on your SQL Server, and then open up the **Jobs** folder. Inside this folder, you will see every job on your SQL Server, whether it is a Maintenance Plan job or not, as shown in Figure 3.20.

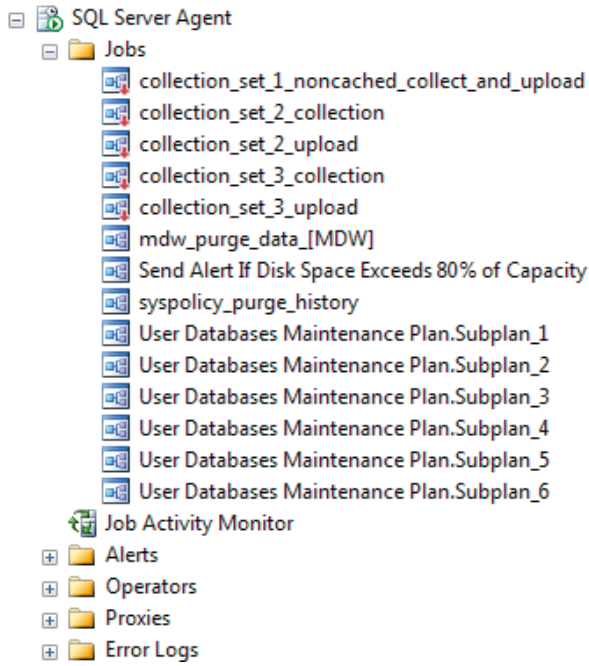

Figure 3.20: The name of your Maintenance Plan will be a part of the SQL Server Agent jobs.

In Figure 3.20, you can see a lot of different jobs, but it is easy to spot the six Maintenance Plan jobs because they include the words "Maintenance Plan" as part of the job name. Each of these jobs will run one of the scheduled tasks that the Maintenance Plan will perform.

Notice that each job has a suffix of "Subplan" along with a number. The term, Subplan, is often used interchangeably with "job" and refers to a scheduled maintenance task; the number of the task matches the order of the tasks you assigned when you created the Maintenance Plan. However, don't forget that the logical ordering of the tasks that you specified is not necessarily the order in which they will be executed. This order is decided by your schedule for each task, as we discussed earlier.

You can double-click on each job to open it up and see what it looks like. While I don't recommend you make any changes, it is interesting to look at the command used to execute the SSIS package for a particular scheduled task, as shown in Figure 3.21.

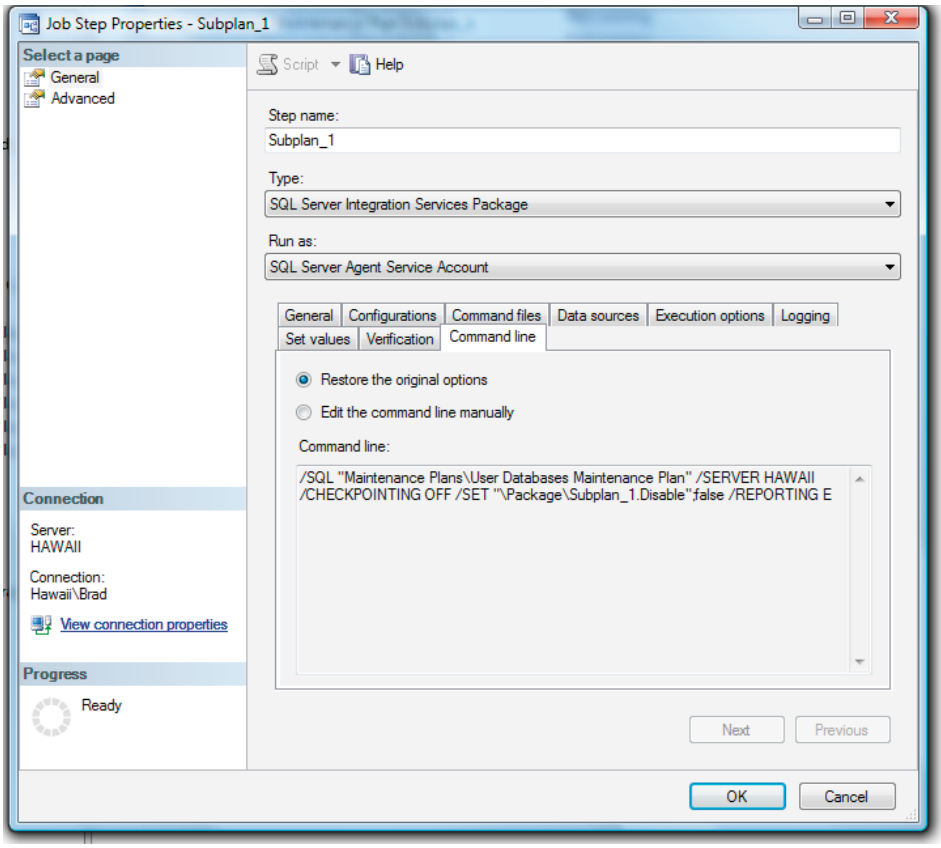

Figure 3.21: This screen shows how the SSIS package will be executed for this maintenance task.

As you can see in this figure, when a scheduled job executes, all it does is execute the related SSIS package, with a number of parameters that specify the plan to be run, the server it is to be run on, and so on. In this particular example, `Subplan_1` of the Maintenance Plan is being run and so is passed in as a parameter.

A word of warning before we move on: while it's possible to customize your Maintenance Plan by directly modifying its SQL Server Agent jobs, I strongly advise against it. Unless you are an expert in both SSIS and the SQL Server Agent, the odds of "breaking" the Maintenance Plan are very high. If you need to make any changes to a Maintenance Plan, use the Maintenance Plan Designer. In this way, any changes will automatically be reflected in the Maintenance Plan's related SQL Server Agent jobs.

Likewise, if you want to delete a Maintenance Plan, be sure you do so right-clicking on the Maintenance Plan's name and clicking **Delete**. If you try to delete the Maintenance Plan's SSIS package or SQL Server Agent jobs directly, you could create a mess that will be difficult to untangle.

Generally speaking, if you feel you need to do a lot of customization to a Maintenance Plan once it has been created, then most likely you would be better off using T-SQL or PowerShell scripts for your database maintenance.

Testing Your Maintenance Plan

Having created a new Maintenance Plan, and before congratulating ourselves on a job well done and going home for the night, we should first test it to see if it runs as we expect.

To test our new Maintenance Plan, we need to run it against SQL Server, preferably during a maintenance window so that any resources used by the plan do not interfere with user activity. So what's the best way to test a Maintenance Plan? At first glance, this seems easy. We can simply right-click on the plan and select **Execute**, as shown in Figure 3.22.

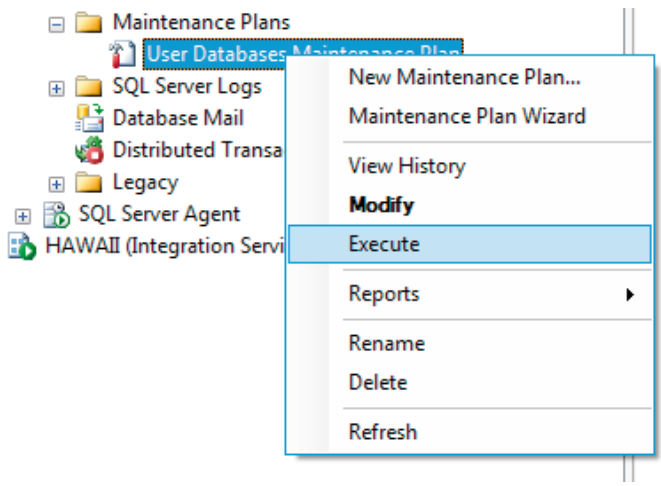

Figure 3.22: Does selecting "Execute" actually execute a Maintenance Plan?

Sounds straightforward enough but, when we select **Execute**, we get the error message shown in Figure 3.23.

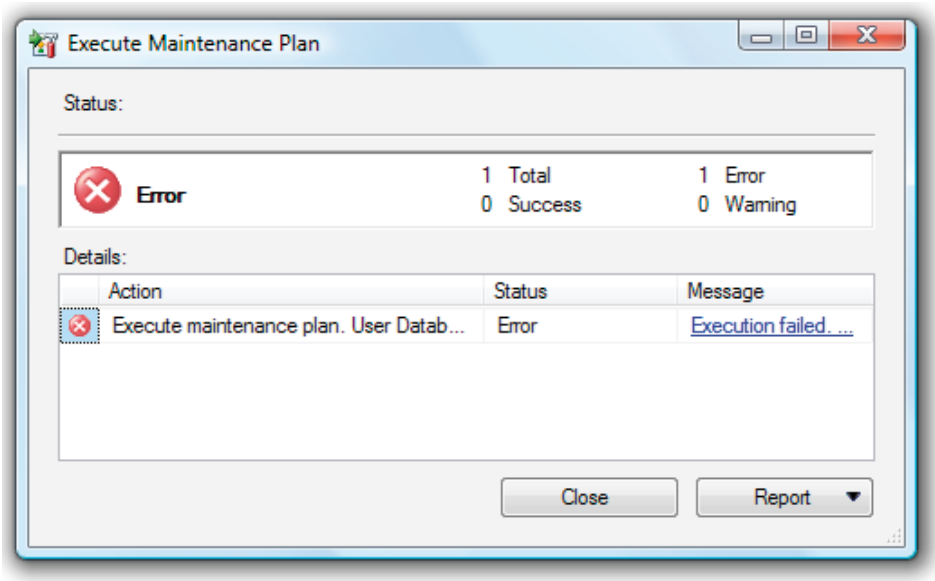

Figure 3.23: Apparently, selecting "Execute" is not a good way to test a Maintenance Plan.

What does this error mean? Is our Maintenance Plan "bad?" In order to answer this question, we can click on the **Execution failed...** link to find out more information about this error, as shown in Figure 3.24

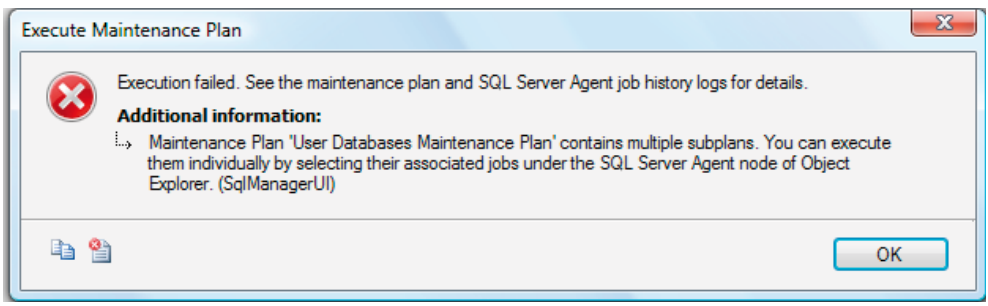

Figure 3.24: Unlike most error messages, this one is actually useful.

The error message is actually useful in this case: "*User Databases Maintenance Plan contains multiple subplans. You can execute them individually by selecting their associated jobs under the SQL Server Agent node of Object Explorer.*"

In other words, the **Execute** option available for a Maintenance Plan will only work if a plan has only one subplan. The topic of subplans is covered in detail in Chapter 18 but for now, just note that a single Maintenance Plan can be made up of several subplans, and each subplan is made up of one or more maintenance tasks. If a Maintenance Plan has more than

one subplan (and most do), then you have to execute the individual SQL Server job for each of the subplans in your Maintenance Plan. When creating our example Maintenance Plan using the Wizard, behind the scenes this was actually implemented as six individual subplans, one for each of the tasks we configured. So, in order to fully test our plan, we will need to manually execute each of the six subplans of the Maintenance Plan in the correct logical order.

Maintenance Task Order

As discussed in the earlier section on Maintenance Task Order, some jobs have to run before other jobs, otherwise they might fail.

So, in order to test our new plan, we navigate to the **SQL Server Agent | Jobs** directory, right-click on the relevant Maintenance Plan Job and select **Start Job at Step...**, as shown in Figure 3.25.

Although this name gives the impression that it might allow us to run all of the jobs, it doesn't. Instead, once we select this option, only the job we have selected will run. Why is this? This is because a "step" is a subset of a job, and all Maintenance Plan jobs have a single "step." The confusion arises because "step" and "job" sound like the same thing, but they aren't. Each job only has a single step, so we must test each job, one after another.

Having run the first job, we run the second, and so on, until they have all completed. Be sure to only run one job at a time, and wait for it to complete successfully before trying the next. If the databases against which the plan is running are large, then these tests can be time consuming, so this extra time must be planned for in advance. In fact, the time it takes for a particular task to run during testing is a valuable data point when determining when the task should be scheduled to run.

Maintenance Task Scheduling*...*

...is covered in full detail in Chapter 4, where you will learn that determining the length of a task is a very important part of creating maintenance task schedules.

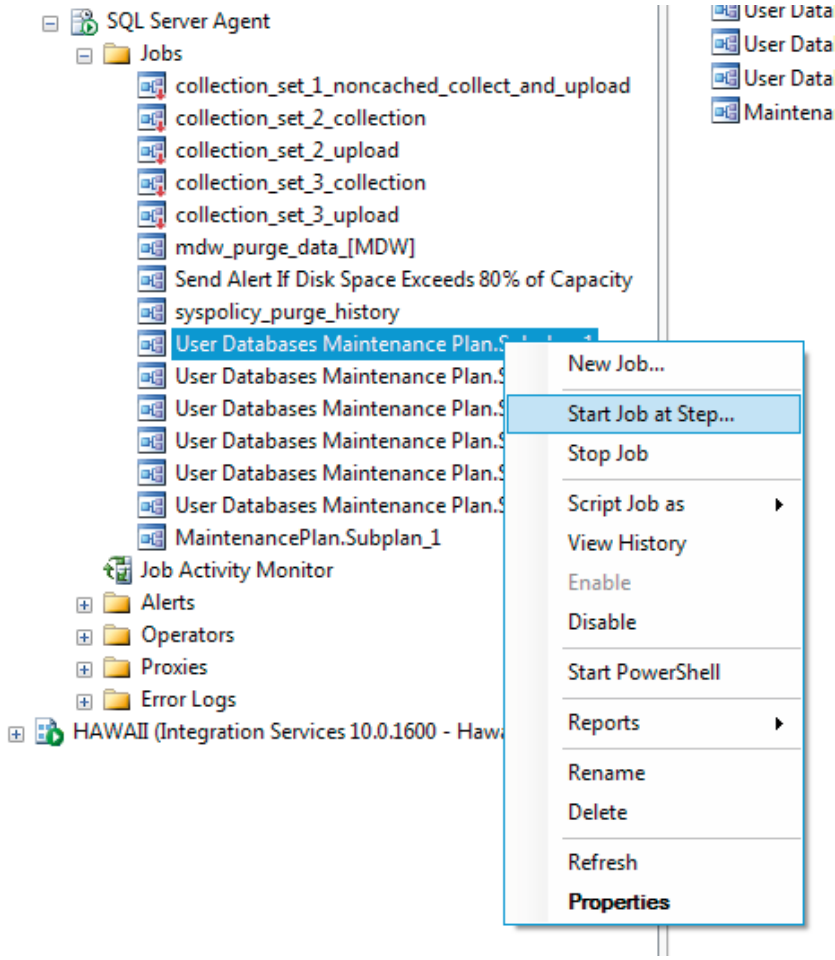

Figure 3.25: You can test each maintenance task by running Start Job at Step...

While all this sounds like a somewhat laborious process, this testing is essential to ensure that your Maintenance Plan, as a whole, will succeed. Furthermore, if a given job fails, you have immediately narrowed your troubleshooting to the investigation of an individual job, rather than the plan as a whole.

Let's go ahead and run the first job in our Maintenance Plan, the `Check Database Integrity` job, and see what happens. Right-click on **User Databases Maintenance Plan. Subplan1**, and select **Start Job at Step**. Hopefully, you'll see a screen similar to that shown in Figure 3.26.

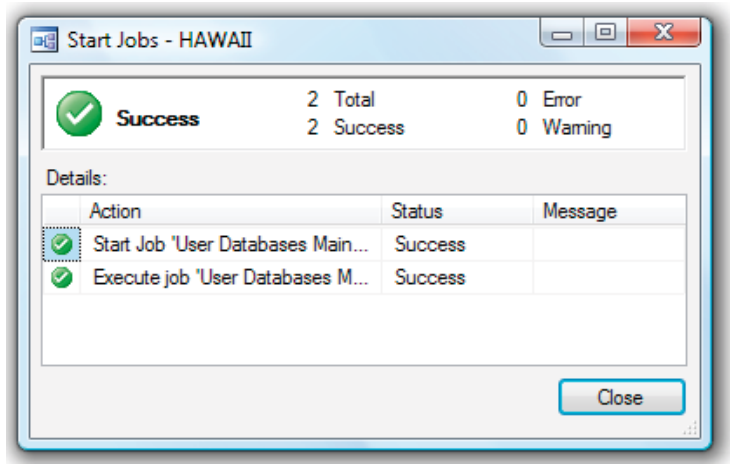

Figure 3.26: The first maintenance task of the Maintenance Plan ran successfully.

Only once you've achieved successful test executions for each of the steps in the Maintenance Plan are you ready to put this Maintenance Plan into production. If you get any errors, be sure to identify and fix the problems before testing the rest of the jobs. Often, if one job fails, then other jobs will fail, and you might as well fix them as soon as you can. Error messages appear as links on the screen, so if you do get an error, check out the error message, and hopefully you will be able to figure out what the problem is and fix it.

So, what's the moral of this story? Don't use a Maintenance Plan's "Execute" option to test plans with multiple subplans. Instead, test one maintenance subplan at a time, using its related SQL Server Agent job.

Summary

Having seen the big picture of how to use the Maintenance Plan Wizard to create a Maintenance Plan, next we will first spend a chapter on job scheduling, then we will spend a chapter on each of the eleven maintenance tasks. This way, you will gain a better understanding of what each one does, and how it best fits (or doesn't fit) with your SQL Server's maintenance needs. As you read about each of these tasks, keep in mind that they apply, not only to the Maintenance Plan Wizard, but also to the Maintenance Plan Designer (which will be discussed later in this book).

Chapter 4: Task Scheduling

Starting with Chapter 5, we begin a run of eleven chapters that explain the ins and outs of each of the eleven maintenance tasks that you can define and schedule as part of a SQL Server Maintenance Plan.

The **scheduling** of a task within a Maintenance Plan, created using the Wizard or Designer, is a step that is a little more complex than it may first appear, and is common to the configuration of every task that we'll discuss in subsequent chapters.

This chapter will therefore provide a broad overview of the essential elements of task scheduling, and the general issues that will determine when, and how often, maintenance tasks should run. This will allow us, in the subsequent chapters, to focus on scheduling advice specific to a given task, rather than the logistics of scheduling.

Scheduling: General Considerations

As part of your overall maintenance plan, some maintenance tasks need to occur monthly, some weekly, some daily, and some hourly. Unfortunately, the Maintenance Plan Wizard offers absolutely no help in deciding which schedule is appropriate for a given task. It does provide some default values, but these values are rarely appropriate for anyone.

What does this mean? It means that, before you create your own Maintenance Plan, you have to consider when, and how often, you want to run the tasks you have selected. In essence, scheduling database maintenance tasks is always a compromise between what you would like to do, and what you have time to do. I can't make this choice for you, but I will try to offer some general advice here, and then, in subsequent chapters, some specific scheduling advice for each of the maintenance tasks covered in this book.

Avoid Scheduling Tasks During Busy Periods

Many maintenance tasks are resource intensive, and can negatively affect the user's experience, especially if you run them during periods where the server is already being heavily used.

In many cases, an organization will specify that such maintenance tasks should be scheduled during specific **maintenance windows**. These are slow periods, often at night or weekends, where few, if any users, are accessing the SQL Server databases. If this is the

case, it is relatively easy to schedule resource-intensive maintenance tasks to run during these maintenance windows. Any maintenance tasks that do not require significant server resources, such as backing up transaction logs, can be scheduled outside these windows, as required.

In other cases, customers will require 24/7 access to the SQL Server databases. If you are faced with such a situation, then you probably shouldn't be using the Maintenance Plan Wizard to create your maintenance plan. Instead, you should be using custom T-SQL or PowerShell scripts that will help you to keep the maintenance task footprint to a minimum.

If your requirements fall between these two extremes, perhaps you need to run a 24/5 shop, a 24/6 shop, or an 18/7 shop, then you'll probably find yourself in the situation of trying to fit many maintenance tasks into little time. If this is the case, you can consider using the Maintenance Plan Wizard, assuming that the plan it creates runs within your available windows. If you can't get your Maintenance Plan to fit within the available windows, then you probably should avoid using the Maintenance Plan Wizard and consider other scripting choices.

Avoid Overlapping Tasks

Certain lightweight tasks can be run simultaneously without overtly affecting server performance. Generally, however, running too many overlapping maintenance tasks is a recipe for a very slow server, or a failed Maintenance Plan. For example, you certainly don't want to perform a `Check Database Integrity` task on a database at the same time as you are running a `Rebuild Index` task, both of which are heavy-duty maintenance tasks.

One of the problems you will find, when you first create a Maintenance Plan with the Maintenance Plan Wizard, is that you won't know how long it takes a particular task to complete. As such, it is fairly easy to make a mistake with the scheduling and end up with overlapping tasks.

Essentially, what you have to do is to make an educated guess as to an appropriate schedule and then observe how long each task takes to run. Be aware, however, that most tasks don't take the same amount of time to run each time they are executed. For example, a `Rebuild Index` task might take 30 minutes one day, and 45 minutes the next day. You may have to record the execution lengths of your tasks over a period of time, in order to determine the longest time it takes to run a task, and then use the highest number as the basis for scheduling jobs so they don't overlap.

Of course, this experimentation may result in some overlapping jobs at first, but you should soon figure out what schedule works best for your server's environment.

There are two different ways to find out how long a maintenance task takes to run. First, you

can go to the task's SQL Server Agent job, right-click on it, and select **View History**. This will display information on when the job started and completed. Second, you can examine the report text file that is created when each task in a Maintenance Plan executes. It includes both the starting and stopping times. In either case, you will need to check the times for all the tasks that are included in your Maintenance Plan.

Task Frequency

In the DBA's dream world, maintenance windows are long and frequent, and users are patient. Most of the tasks in the Maintenance Plan Wizard can be scheduled to run in the daily maintenance window, with the exception of the `Backup Database (Transaction Log)` task, which runs hourly. All servers are well and frequently maintained, and the DBA leads a tranquil, unhurried existence.

Back in the real world, however, task scheduling is always a compromise. Maintenance windows are rarely long enough to perform all of the tasks exactly when you'd like to perform them, and users are intolerant of server slow-downs. If such cases, the DBA must juggle the tasks as best he or she can, fitting in the most resource-intensive ones at times that will minimize their impact.

For example, let's say that the only maintenance window is Sunday. If that is the case, you will still want to take full, daily backups and hourly transaction log backups, as they are essential and don't have a huge negative performance impact. However, the `Rebuild Index` and the `Check Database Integrity` tasks are both very resource intensive and so you may only be able to run them once a week, during the Sunday maintenance window. Of course, this is a compromise because, ideally, these tasks would be run daily for optimum performance and high availability.

Task Scheduling in the Wizard

In order to follow along with this chapter, you'll need to start the Maintenance Plan Wizard, as described in the *Starting the Maintenance Plan Wizard* section of Chapter 3. When you reach the **Select Maintenance Tasks** screen you can, for the sake of this discussion, select any one of the available tasks. I've chosen the first one: Database Check Integrity. Click **Next** to confirm your selection and then **Next** again to get past the **Task Order** screen. You will then reach the **Define Database Check Integrity Task** screen, shown in Figure 4.1.

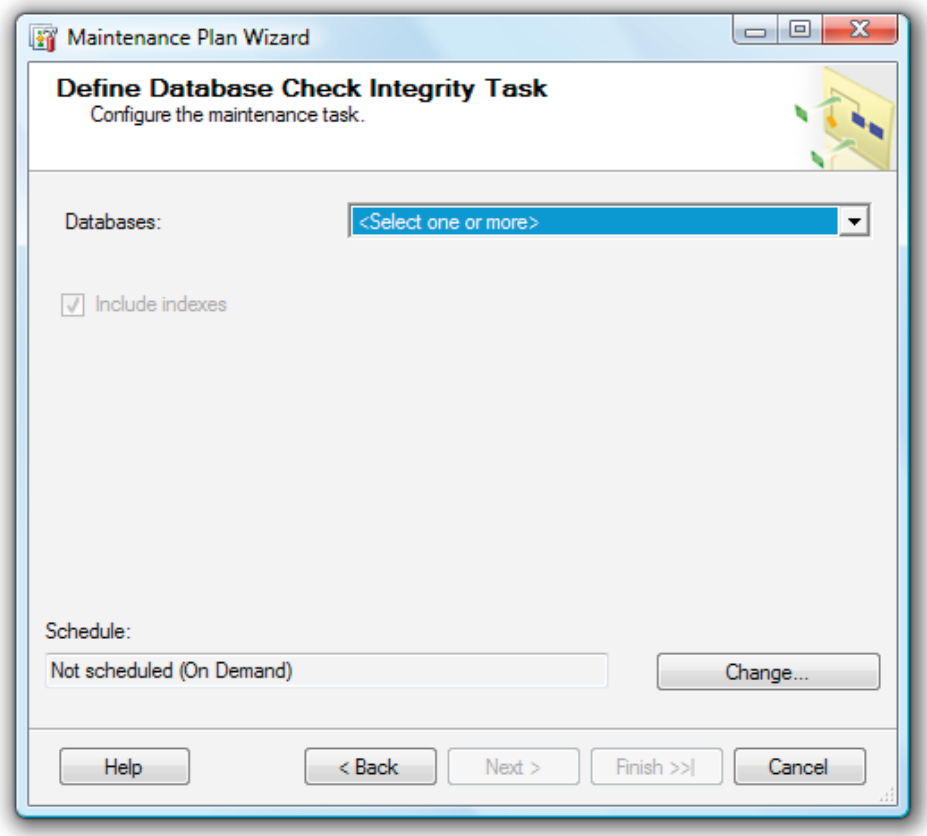

Figure 4.1: Configuring maintenance tasks – the Schedule option.

At the bottom section of the screen shown in Figure 4.1, we see the **Schedule** option for our Maintenance Plans. This option will appear on every task where **Separate schedules for each task** option was specified for the Maintenance Plan.

The default selection for **Schedule** is **Not scheduled (On Demand)**, as shown in Figure 4.2.

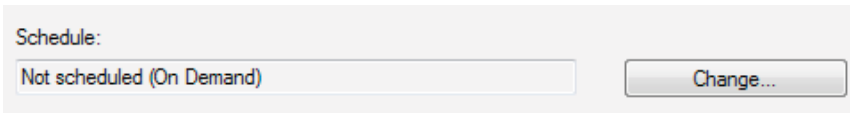

Figure 4.2: By default, tasks are not scheduled, so you need to choose a schedule for each task in the Maintenance Plan Wizard.

If we accept this default, and don't set a schedule for a given task, then when the Maintenance Plan Wizard is complete, the task will be created, but a scheduled job for the

task will not be created. This means the task will not run automatically and we'll need to run it manually instead.

> ***Scheduling using the Maintenance Plan Designer***
>
> *If you create an "on demand" task, you can always add a schedule at a later time using the Maintenance Plan Designer.*

Since our goal, generally, is to automate our maintenance tasks, not to perform them manually, we'll be creating a schedule for the tasks covered in this book.

Job Schedule Properties

When a task is assigned to a schedule, that task is referred to as a "job," in light of the fact that the task will be executed as a job from within the SQL Server Agent. In this section, we'll discuss the logistics of creating a job schedule. In order to begin this process, click on the **Change** button shown in Figure 4.1, and the **Job Schedule Properties** screen, shown in Figure 4.3, will appear.

Let's take a closer look at each section of this screen, starting at the top.

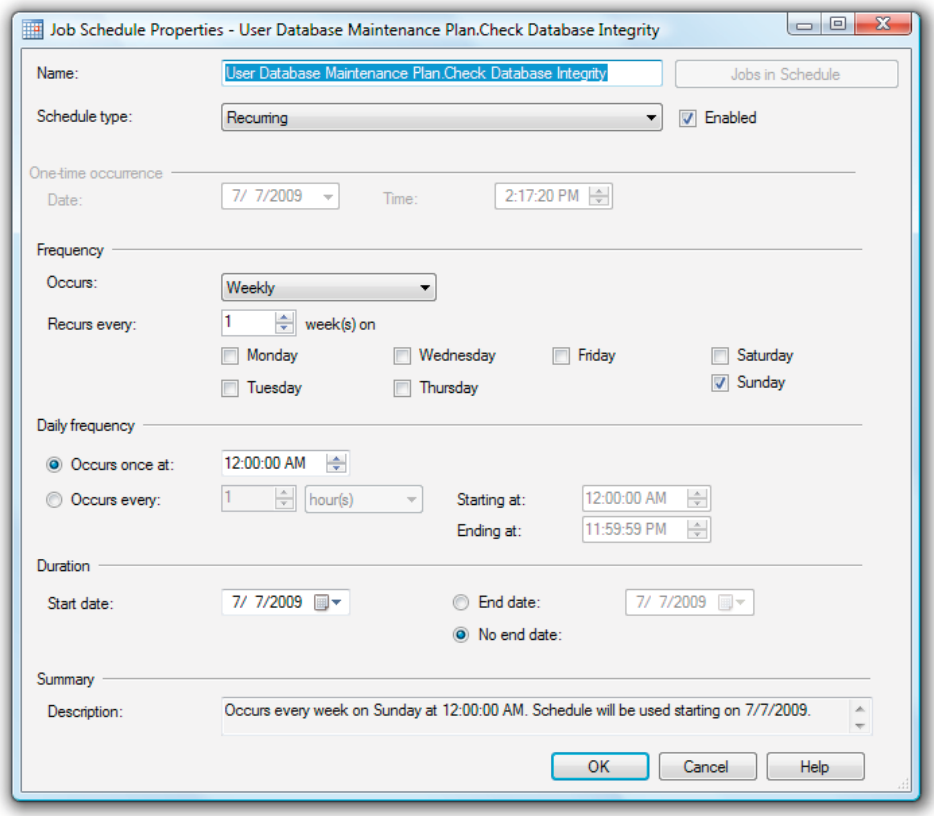

Figure 4.3: The "Job Schedule Properties" screen can be a little overwhelming.

Job Schedule Name and Type

The first thing you will notice is that the name of this job schedule has been filled out for you, using a combination of the Maintenance Plan name and the name of the task being scheduled. In this example, it is "User Database Maintenance Plan.Check Database Integrity," as shown in Figure 4.4.

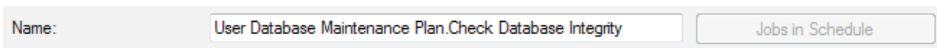

Figure 4.4: Notice that this schedule has been given a default name.

I don't see any reason to change this name, as the naming scheme makes a lot of sense. Next to the schedule name is a grayed out box, titled **Jobs in Schedule**. This feature is not relevant to Maintenance Plan scheduling, and you can ignore it.

The redundant Jobs in Schedule button

You're probably wondering why, if the button is not used, it appears on the screen. This particular screen is shared code that is used by other parts of SSMS. The developers of SSMS decided to reuse the same scheduling code instead of creating additional code, specifically designed for the Maintenance Plan Wizard. The result is a button we don't need, along with a few other options we don't need either, as we will soon see.

In the next part of the screen, you must specify the **Schedule type**. The default option is **Recurring**, as shown in Figure 4.5, and is the one you should choose, as the goal of a Maintenance Plan is to create recurring jobs that run specific tasks.

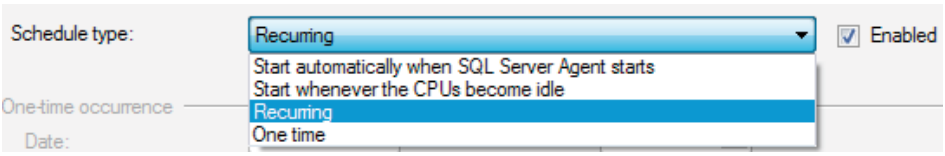

Figure 4.5: You have to specify when the job is to occur, and whether it is enabled or not.

The other options, while they could be used, are really designed for other purposes (remember, this is shared code) and not for the Maintenance Plan Wizard. In fact, using any of the options other than **Recurring** could easily get you into trouble, as they are not time based, and thus you cannot schedule a specific time to run a task.

By default, the **Enabled** checkbox, to the right of **Schedule type**, is selected indicating that the job schedule is enabled and active. You should accept this default and leave the box selected. If, for some reason, you want to temporarily turn off a scheduled task after a Plan has been created, you can do so using the Maintenance Plan Designer, which is discussed in Chapter 16 and onwards.

Job Frequency

The next part of the **Job Schedule Properties** screen is called **Frequency**, as shown in Figure 4.6. The appearance of this screen varies depending on the option selected for **Occurs**. By default, this is **Weekly**, which is a little deceiving. When I think of weekly, I think of once a week, but that is not what this option means. It means that you have the option to select which days in a week you want a job to run. Only Sunday is selected by default, but you can choose any day of the week, and as many of them as you want.

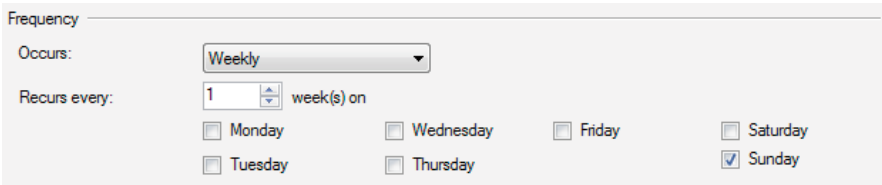

Figure 4.6: You must choose how often a task is to be executed.

In some cases, you may not want to perform a task on a weekly basis, but every two weeks, or every four weeks. If this is the case, you can change this behavior by setting the value for **Recurs every**. The default value is 1 (weekly) but you can increase it to 2 for every two weeks, and so on. In most cases, you will find it simplest to use 1 as the **Recurs every** value; otherwise it gets hard to keep track of your schedule.

If you choose **Daily**, instead of **Weekly**, for **Occurs** then you'll notice that the screen changes slightly in appearance, as shown in Figure 4.7.

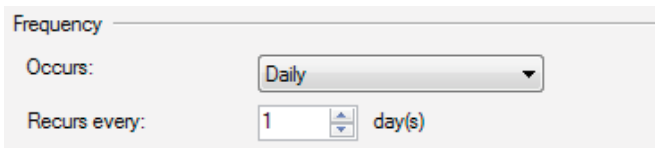

Figure 4.7: The Daily option allows you to select how often daily really is.

The **Daily** option also gives the impression that the job is to run once a day, but that is not the case. In fact, you can schedule jobs to run as often as want, on a daily basis, as you'll see when we reach the **Daily Frequency** section of the screen, shortly.

The **Recurs every** option works just as described previously. The default value of 1 indicates that the job will run every day; a value of 2 means every other day, and so on. Again, I recommend you keep it simple and leave **Recurs every** at its default value of 1.

If you choose the final option, **Monthly**, the screen changes once again, as shown in Figure 4.8.

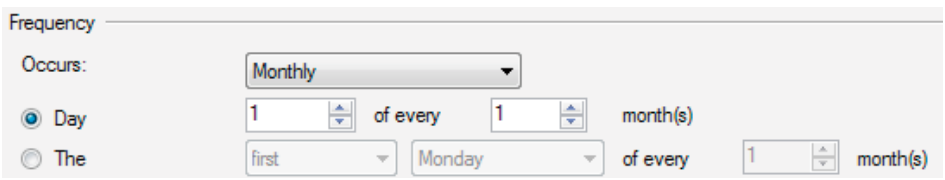

Figure 4.8: The Monthly options gives you more options that you can probably find good uses for.

The **Day** option has two configurable values. The first value refers to the day of a month. For example, if the value is 1, then the job will occur on the first day of every month, if the value is 2, it would mean that job occurs on Day 2 of every month, and so on. The second value refers to the month. A value of 1 means the job is to run every month, if the value is 2, then the job runs every other month, and so on. This option can get complicated very quickly.

Rather than use the **Day** option, you can, instead, use the **The** option to specify job frequency, as shown in Figure 4.9.

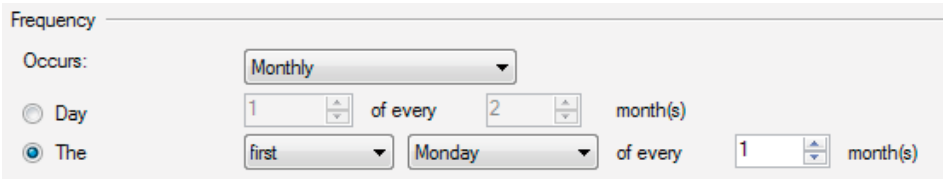

Figure 4.9: The "The" option gives you even more choices.

The default setting specifies that the job will occur on the first Monday of every month. If you change "first" to "second," this would mean the second Monday of every month, and so on. You can also change the day of the week, and for which months. Again, all of this gets complicated very quickly, and I suggest you keep things as simple as possible, and avoid using this option.

Daily Frequency

The next option on the **Job Schedule Properties** screen is **Daily Frequency**, as shown in Figure 4.10.

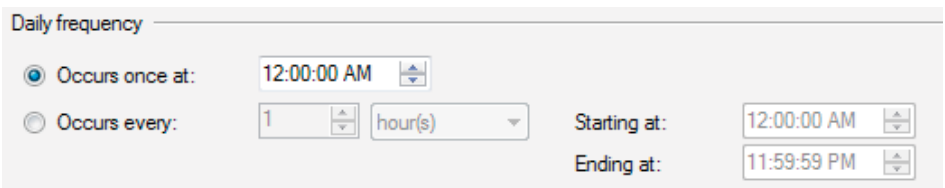

Figure 4.10: The Daily frequency option allows us to schedule a job to occur more than once a day.

Didn't we just see a daily option in the **Frequency** section of the screen? Yes, we did, but this **Daily frequency** option means something entirely different. It refers to when and how often on the day(s) selected in the **Frequency** section the task will occur. The **Occurs once at** option allows us to schedule the time of a day the task is to run once. By default, a task is scheduled to occur once a day at 12 a.m.

If you want a task to run more than once a day, then you must select the **Occurs every** option, as shown in Figure 4.11.

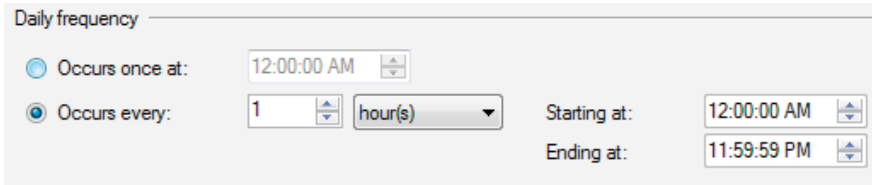

Figure 4.11: Events can be scheduled to occur multiple times a day.

By default, when you choose this option, a job is to be executed every 1 hour. You can change the frequency the job runs, along with the time scale (hours, minutes, and seconds). For example, you could schedule a job to run every 12 hours, every 12 minutes, or every 12 seconds.

If this is not quite enough flexibility, you have the option to control the time period within a day that jobs can run. The default value is a **Starting at** value of 12:00:00 a.m. and an **Ending at** value of 11:59:59 p.m., which is one second less than a full 24-hour day. This means that your job can run any time during a 24-hour day.

If you want to prevent jobs from running during certain parts of the day, simply change the **Starting at** and **Ending at** times. For example, you might decide to restrict certain tasks to nighttime execution so as not to interfere with user activity.

Job Duration

The final choice you can make is to specify when your job schedule is to begin and, optionally, to end, using the **Duration** option. The default, shown in Figure 4.12, is to start the job schedule on today's date, with no end date.

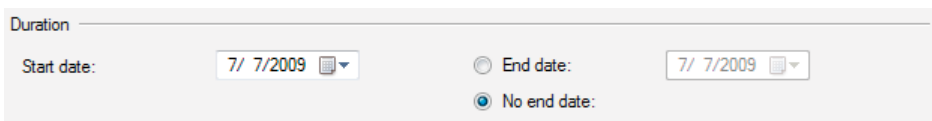

Figure 4.12: You can determine when a job starts and ends.

Alternatively, you can specify the job schedule to begin at some future date, or to end the job schedule at some future date. In the context of the Maintenance Plan Wizard, you would rarely want to change either of these defaults, as your goal is to start your jobs right away, and have them run forever (or until you decide to change or delete the job). The one exception I can think of is that you might want to delay a job from starting right now, until a later date, in order to ensure that jobs that are dependent on one another occur in the correct order.

For example, you have to schedule at least one full backup before you schedule a transaction log backup.

Scheduling Individual Maintenance Tasks

Hopefully, the previous sections have fully explained the options for scheduling the various maintenance tasks that will make up your Maintenance Plans. The exact options you choose will depend on:

- **The specific task that you are scheduling** – tasks such as `Backup Database (Transaction Log)` will be scheduled hourly; tasks such as `Backup Database (Full)` will be daily; and tasks such as `Rebuild Index` might be weekly.

- **General scheduling considerations** – as discussed earlier in this chapter, you will need to schedule as best you can to maximize use of maintenance windows, and avoid overlapping tasks, which could impact server performance.

Over the coming chapters, we'll discuss specific scheduling considerations for each individual task. Throughout my explanation of the Maintenance Plan Wizard, I am going to assume that I only have a single weekly maintenance window, which is the entire day of Sunday.

Summary

As I forewarned at the start of this chapter, scheduling your maintenance tasks is a more complex process that you may at first think, and needs careful consideration.

The **Job Schedule Properties** screen doesn't really help matters by offering more options and flexibility than you really need. My general advice is to plan carefully, make the best possible use that you can of the available maintenance windows, and keep your scheduling as simple as possible.

We now move in to discuss in full detail the specific maintenance tasks that can be created and scheduled using the Wizard, where I'll offer more specific advice on the most appropriate schedule for a given task.

Chapter 5: Check Database Integrity Task

Starting with this chapter, we begin an eleven chapter section on the various maintenance task options available when creating Maintenance Plans using the Wizard. In this chapter, we learn about the `Check Database Integrity` task, what it does, and how to configure it. The purpose of the `Check Database Integrity` task is to check the logical and physical integrity of all the objects in a database, looking for any corruption that could put your data at risk. The goal of the task is to try and identify small integrity problems early, so that they don't become bigger problems later.

An Overview of the Check Database Integrity Task

With modern servers, data file corruption on storage devices is not as common as it used to be. In fact, many people have become rather complacent about potential data corruption, assuming that it will never happen to them. Hopefully, you will be lucky and never experience it, but if it does happen, it can be a nasty surprise.

What can be really surprising is that a data file, such as a SQL Server `MDF` or `LDF` file, might become corrupt, but you may not know about it right away. Many people assume that, if data corruption does occur, it will show its ugly head immediately. But that is not always the case. It is possible that one or more of the sectors that hold part of a file can become corrupted, but SQL Server won't notice it until long after the corruption has occurred. In other words, a database's `MDF` or `LDF` file might actually have corruption, but you may not find out about it until days or weeks later. This is not common, but it can happen.

In the worst cases of data corruption, it is possible that your only option to fix the corruption is to restore a backup. However, if you do not spot the corruption immediately, then it's possible that the corrupted data will make its way into your backups. If it has, then you'll find that you won't be able to use that backup.

If the most recent "good" backup is, say, a week old, that means you will have to restore a database that is a week old, and you will possibly have lost all the modifications that have been made to the database since that last good backup.

If you want to be a diligent DBA, it is your responsibility to regularly verify that your databases don't have any corruption. Fortunately, SQL Server provides a built-in command, DBCC CHECKDB, for just this purpose. When this command is run on a database, it checks both the logical and physical integrity of all objects in that database. If the command finds any problems, it will display them in a report, and at that point you have to figure out what to do next, a topic that is beyond the scope of this book. While I can't tell you what to do in a particular case of corruption, I can tell you that the sooner you find it, the better off you will be.

> ***Dealing with data corruption***
>
> *For more information on how to deal with database corruption, see the article "Help, my database is corrupt. Now what?," by Gail Shaw, at* http://www.sqlservercentral.com/articles/65804/*, or "Finding Data Corruption," by Rodney Landrum, at* http://www.simple-talk.com/sql/database-administration/finding-data-corruption/*.*

The DBCC CHECKDB command has numerous optional parameters that control such things as the comprehensiveness of the check, whether or not informational messages on the progress of the command should be displayed, and how many error messages should be displayed; it even has some repair options.

> ***The DBCC CHECKDB Repair options***
>
> *If you read about the* DBCC CHECKDB *command in Books Online, you may notice that if offers some "repair" options. Unless you know exactly what you are doing, don't use these options, as you may end up causing more damage to your database. If you don't know how to resolve database integrity problems, you need to consult an expert before proceeding.*

Within the context of the Maintenance Plan Wizard, the DBCC CHECKDB command is executed under the name of the `Check Database Integrity` task. When you configure this task to run using the Wizard's default settings, the following command runs against the databases you select.

```
DBCC CHECKDB('database_name')WITH NO_INFOMSGS
```

When this command is run, all DBCC CHECKDB tests are performed, informational messages (status messages telling you what is being checked) are not returned, and up to a maximum of 200 error messages are returned. If more than 200 error messages occur, only the first 200 will be included in the report text file produced when this task is run.

Controlling DBCC CHECKDB

As discussed previously, DBCC CHECKDB *has many options that govern how it performs, and most of these options cannot be controlled from the Maintenance Plan Wizard. If you need finer-grained control over this command, you will need to use T-SQL or PowerShell scripts.*

If you review the report from this task, and no error messages are returned, it looks something like the following:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: User Databases Maintenance Plan
Duration: 00:00:04
Status: Succeeded.
Details:
Check Database Integrity (HAWAII)
Check Database integrity on Local server connection
Databases: AdventureWorks
Include indexes
Task start: 2009-07-29T10:12:36.
Task end: 2009-07-29T10:12:41.
Success
Command:
USE [AdventureWorks]
GO
DBCC CHECKDB(N''AdventureWorks'')  WITH NO_INFOMSGS
GO
```

As you can see, the report reveals some basic information about the task, such as its duration, its success, the database checked, the time it took to run, and the actual command that was run. Also, notice that the report ends immediately after the command. In other words, the DBCC CHECKDB command did not find any errors, so there are no errors to report. If there had been error messages, then they would be listed at the bottom of the report.

If you do get any error messages, I would suggest that you run the DBCC CHECKDB command, manually, using the following T-SQL command.

```
DBCC CHECKDB ('DATABASE_NAME') WITH NO_INFOMSGS, ALL_ERRORMSGS
```

What this command does is to list all the error messages produced by running the command, not just the first 200 of them. If your database is having corruption problems, you want to know about all of them, not just the first 200. You can then read through the messages and try to identify what the problem is, and figure out what to do next, which often, unfortunately, is to restore the last known good database.

When and How Often to Run Integrity Checks

As a DBA, you do not want to allow corruption to creep unnoticed into your databases, so it is essential that the `Check Database Integrity` task is scheduled to run on a regular basis. At the same time, it is also important to keep in mind that this task is very resource intensive, and running it at the wrong time can have detrimental effects on your SQL Server's performance. You will want to treat it as an "offline" task, to be performed when as few people as possible might be accessing the database. Generally speaking, this means during a scheduled maintenance window.

Here are my specific recommendations on when, and how often, the `Check Database Integrity` task should be run.

- If you have a nightly maintenance window that allows you to run the `Check Database Integrity` task daily, then do so, as you want to discover any data integrity problems as soon as possible.
- If you can't perform this task nightly, then, at a minimum, it should be run once a week.
- If you don't have a maintenance window long enough to run this task at least once a week, then you probably shouldn't be using the Maintenance Plan Wizard for your maintenance plans. T-SQL or PowerShell scripts offer greater flexibility.

Configuring the Task

Now that we know what this task does, let's see how to configure it using the Maintenance Plan Wizard. The first option on the **Define Database Check Integrity Task** screen is the **Databases** drop-down box. As described in Chapter 3, choose the **All user databases** option and click **OK**. This means that the task will run on all current user databases on this SQL Server instance, and that any new user databases that are subsequently created will automatically be added to the list and subject to the task. The resulting screen will look as shown in Figure 5.1

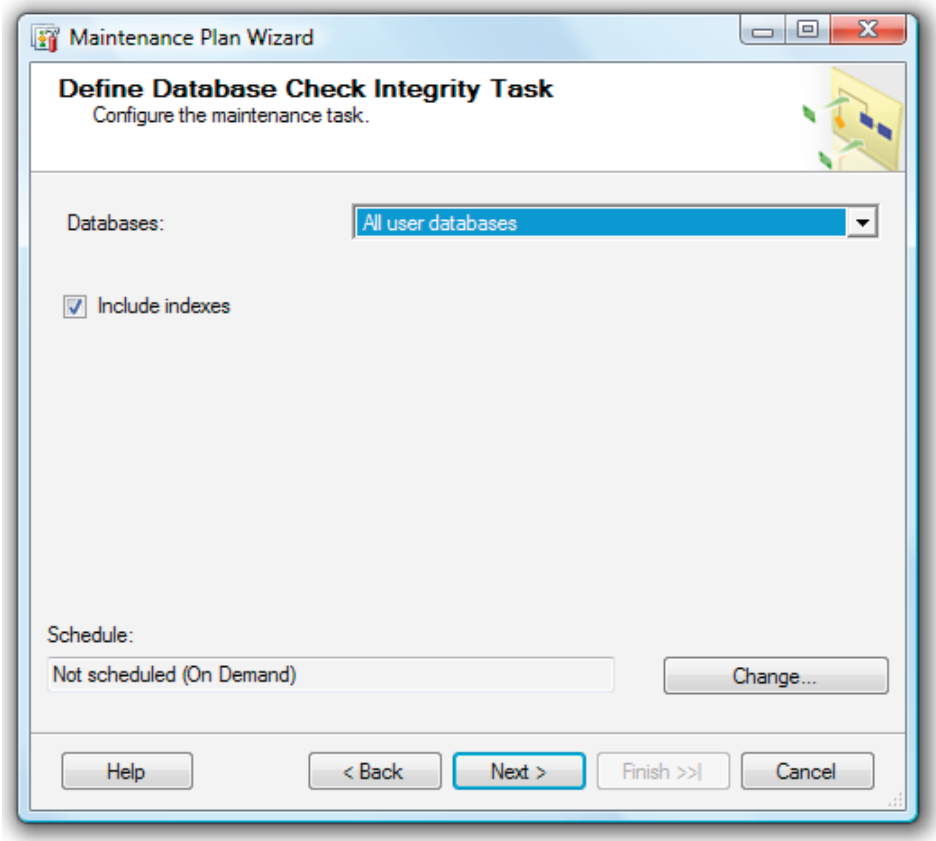

Figure 5.1: Configuring the Check Database Integrity Task.

The "Include indexes" Option

The only option that is specific to this task is the **Include indexes** option, which you will notice is selected by default (see Figure 5.1). When this option is selected, it means that the task will perform the integrity checks against all tables without indexes, plus all tables with clustered and non-clustered indexes.

As discussed earlier, the `Check Database Integrity` Task is resource intensive. If you run this task against very large databases, it can hurt the server's overall performance for the duration of the task. If you have an available maintenance window to perform this task, then running it with the default options is not a problem. However, what if the maintenance window is too short to allow you to run the default command DBCC command for all of your databases? You still need to perform this task, and there is one way to help reduce the performance impact, and that is by deselecting the **Include indexes** checkbox.

When you do this, the `DBCC CHECKDB` command is run using the `NOINDEX` option, as follows:

```
DBCC CHECKDB('database_name', NOINDEX)
```

This means that resource intensive checks of non-clustered indexes are omitted, which reduces the load on the server and shortens the amount of time the task takes to run.

What do you give up by not checking non-clustered indexes? Not a lot, as all the clustered indexes and heaps that hold your data are checked. Of course, by omitting non-clustered indexes from the check, you run the risk of missing potential data integrity problems in these indexes. However, some DBAs regard this as an acceptable risk since you can generally solve the problem of corruption found in a non-clustered index by simply dropping and re-adding the index.

If you have a long enough maintenance window, I suggest you leave the **Include indexes** option selected, and only deselect it if your maintenance window is not long enough for the full integrity check to complete.

Creating the Job Schedule

At the bottom section of this screen, you are invited to create a schedule for our task. Click on the **Change** button, shown in Figure 5.1, to bring up the job scheduling screen. A full description of all the options available on this screen is provided in Chapter 4.

With the previous advice in mind, how would I schedule the `Check Database Integrity` task? If I make the assumptions that I have the entire day of Sunday as my maintenance window, then I would choose the options shown in Figure 5.2.

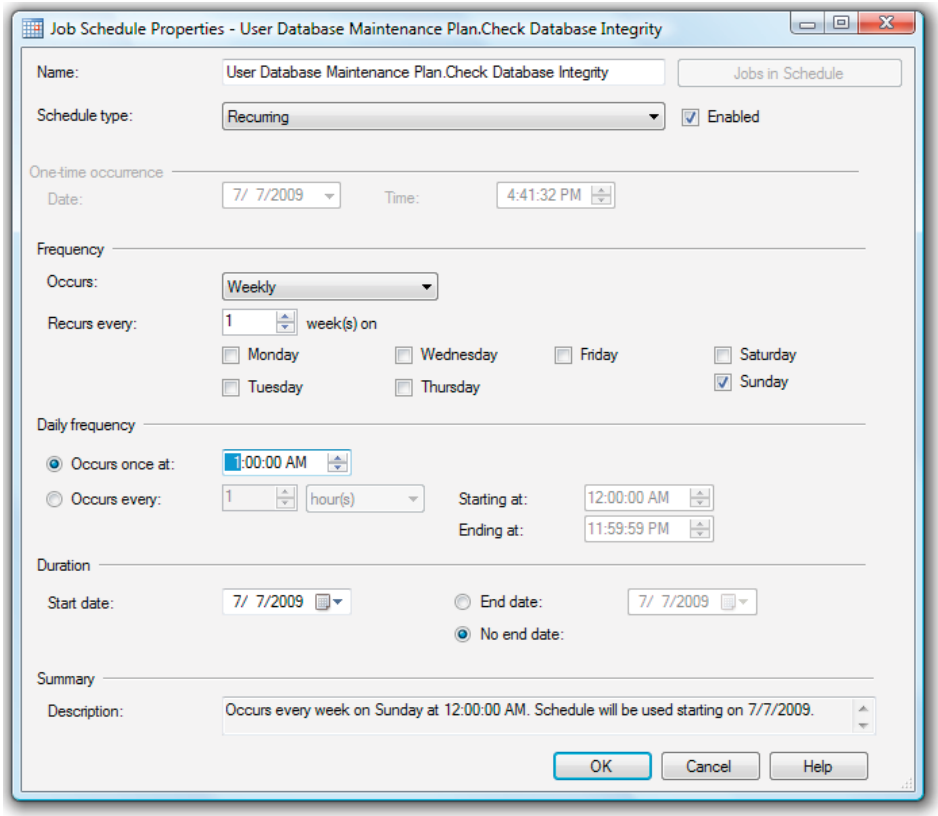

Figure 5.2: This is one way you might want to schedule your Check Database Integrity task.

As you can see, I have kept my schedule options simple. I will run the job once a week, on Sunday, starting at 1 a.m. I arbitrarily chose 1 a.m. as my starting time, but you can choose what time best suits your available maintenance window.

Once you are done with the schedule, click on **OK**. The **Define Database Check Integrity Task** screen reappears and now has a schedule at the bottom of the screen, as shown in Figure 5.3

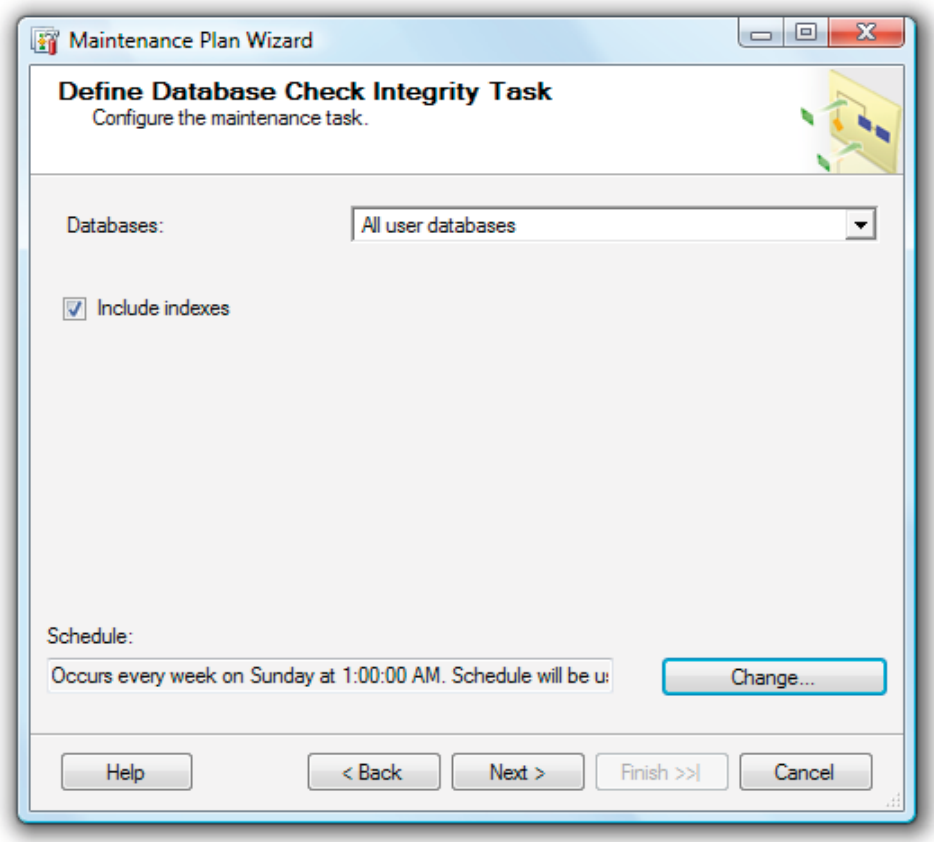

Figure 5.3: Part of the schedule can be seen from this screen of the Wizard.

We are now done with this part of the Wizard, and our `Check Database Integrity` task is created and scheduled to run on all user databases.

Summary

It is very important to run the `Database Check Integrity` task on all your SQL Servers, as it is the only way to definitely know if your databases and your backups are in good shape or not. Don't take the attitude that, just because it is rare for a database to become corrupt, you can skip this maintenance task. While database corruption is rare, when it does occur, often the only way to fix the problem is to restore your database. To ensure that your database doesn't contain any corruption, you should run the `Database Check Integrity` task as often as is practical.

Chapter 6: Shrink Database Task

In Chapter 3, I recommended that you never use the `Shrink Database` task, and that advice stands firm. In this brief chapter, I will describe exactly what is meant by "shrinking a database," why I advise against doing it using the `Shrink Database` task in the Wizard. I will also explain the legitimate reasons to shrink your databases, and the right ways to do it.

Sizing Your Database Files

A database is composed of at least two physical files: one `MDF` (data) file where data is stored, and one `LDF` (log) file, where the transaction log is located. A database can actually have more than two physical files but, for the sake of this discussion, we'll assume it comprises one `MDF` file and one `LDF` file.

When a new database is created using default settings, the initial size of the `MDF` and `LDF` files will be small, their autogrowth setting will be turned on, and file growth will be set to **unrestricted**. Using these default values is unsuitable for many databases. In busy databases, these files can grow rapidly and be subject to frequent autogrowth events. These events are resource-intensive and can have a dramatic impact on the performance of your server when they occur.

As well as inducing potentially unnecessary autogrowth events, this incremental file growth can cause other issues, such as increasing physical file fragmentation, and the creation of excessive virtual log files within the log file.

A full discussion of these issues, and of how to correctly size your data and log files, is beyond the scope of this book, but the salient point here is that it is a recommended best practice to **pre-size** the physical data and log files to their estimated, future production sizes. In other words, when you create a database, you should size these files, not only so that they can cope with the current volume of data, but also so that they can accommodate predicted future growth (for the coming year, for example). The autogrowth feature should be left activated, but the DBA should not *rely* on it to grow their data and log files.

As time passes, and more data is added to a database, more and more of the `MDF` space will be used. At some point, the DBA will want to manually expand the database, in a controlled fashion, to ensure that there is always room for new data, without having to rely on an automatic autogrowth event to kick in and grow the database automatically.

Problems with the Shrink Database Task

If you size your database files correctly to accommodate predicted growth, you will initially have a lot of unused space in your data and log files. For example, let's say that you have created a physical `MDF` file of 100 GB, which you have guesstimated will be large enough to hold all of the data added to it for the upcoming year. After the database has been in production one month, only 10 GB of the file is being used to store data. This means that there is 90 GB of `MDF` space that has been reserved, but not used. This is a normal situation and means that the DBA's growth prediction was more or less accurate.

When you **shrink** a database, what you are doing is reducing the physical size of its files, by reducing the amount of **reserved space** in the file. Shrinking a database never shrinks the data in a database; it only reduces the unused space in a database. This is what the `Shrink Database` task in the Maintenance Plan Wizard does: on a scheduled basis, it arbitrarily removes unused space from database files.

Shrinking a database without a specific goal in mind can cause problems, the most obvious one being that it can remove the space that was intentionally pre-allocated when the DBA first created the database. Our DBA, who has diligently and correctly sized his `MDF` file to 100 GB, may find that an ill-conceived `Shrink Database` task has come along and reduced the file in size so there is no longer enough room to hold the data that is expected to be added to the database within the next year.

Shrinking a database can also reduce the performance of SQL Server. Firstly, the act of shrinking the physical files is resource intensive. Secondly, the shrinking process contributes substantially to index fragmentation. If you shrink a database, but forget to rebuild its indexes immediately thereafter, you risk a performance hit due to heavily fragmented indexes.

If your database does rely on autogrowth events to handle increasing data volume in the file, you may find that use of the `Shrink Database` task gets you into a nasty and expensive grow-shrink cycle. Depending on how the `Shrink Database` task is configured, you can end up in a situation where the database grows throughout the day as new data is added, new indexes are built, or old indexes are rebuilt, and then at night, any excess space beyond what is required to store the data is removed. This means that, the next day, there is not enough room in the database to add more data, create new indexes, or rebuild old indexes, and the database has to use autogrowth again in order to allocate space to perform these tasks. That night, any extra space is removed again, and so on. This cycle uses up a lot of resources, causes index fragmentation, and even physical file fragmentation.

In short, shrinking a database arbitrarily can cause many problems and should be avoided. But there are legitimate reasons, and proper ways to do it, as we discuss next.

The Auto Shrink Option

If you pull up the properties window for a database in Management Studio, and look in the Options page, you'll see an option called `Auto Shrink`*. This option is turned off by default, and should remain turned off. Like the* `Shrink Database` *task, this database option periodically takes a look at empty space in a database, and shrinks it if it exceeds a specified amount. This causes the same problems as those discussed for the* `Shrink Database` *task, and should not be used.*

The Right Way to Shrink a Database

Let's return to our example of the diligent DBA, attempting to correctly pre-size the physical database files. But let's say that the DBA made a mistake, and that the 100 GB estimate for the `MDF` file was way too high. After a year, the actual data stored in the `MDF` was only 15 GB, not the 100 GB that was predicted.

In this situation, especially if disk space is limited, the DBA might choose to shrink the database to a smaller size, say to 50 GB. This would be a perfectly good choice, and the DBA could use the appropriate `DBCC SHRINKDATABASE` or `DBCC SHRINKFILE` command to manually reduce the 100 GB `MDF` file to 50 GB. This procedure is resource intensive, and should be scheduled to run during an available maintenance period. Again, because the shrinking process contributes to index fragmentation, the database should have its indexes rebuilt immediately after shrinking to ensure optimal performance.

When done for a specific purpose, and shrunk using the steps described above, shrinking a database is not a problem.

Summary

If a DBA has a legitimate need to shrink a database, and follows the steps to shrink it properly, then it is as valid as any other maintenance task that a DBA has to do.

Problems arise when DBAs shrink a database without knowing why they are shrinking it, and this is what happens when the `Shrink Database` task is used. It causes unnecessary stress on SQL Server, which can result in serious performance problems.

So, one last time in case you missed it: don't use the `Shrink Database` task as part of your Maintenance Plans. In fact, I'm not even going to show you how to use it.

Chapter 7: Rebuild Index Task

This chapter will describe how to use the `Rebuild Index` task in the Database Maintenance Wizard to maintain the health of your database indexes which, in turn, can boost the performance of your queries. It will cover:

- what `Rebuild Index` does and the problems that can arise if it is not used
- considerations when using the task, and possible alternatives
- how to configure and schedule the task using the Wizard.

An Overview of the Rebuild Index Task

You will recall from Chapter 2 that, as indexes are subjected to data modifications, index fragmentation can occur in the form of **gaps in data pages**, which creates wasted empty space, and **logical fragmentation**, a logical ordering of the data that no longer matches its physical ordering.

Gaps in data pages can reduce the number of rows that can be stored in SQL Server's data cache, leading to increased disk I/O. Logical fragmentation can cause extra disk activity, as the disk subsystem has to work harder to find the data on disk and move it to the data cache. The only way to remove wasted space and logical fragmentation is to rebuild or reorganize the indexes on a regular basis. This is one of the most useful and powerful maintenance tasks that you can perform on a database, because the steps it performs can greatly boost database performance.

If you configure the `Rebuild Index` task using all the default settings, as we did in Chapter 3, when the task runs, it physically drops and rebuilds every index in your selected databases, removing both wasted empty space and logical fragmentation. As a byproduct of rebuilding all the indexes, index and column statistics are also recreated anew and fully updated.

The T-SQL command that is generated from these default settings is as follows:

```
ALTER INDEX index_name ON table_name REBUILD PARTITION = ALL WITH
( PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, ALLOW_ROW_
LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON, ONLINE = OFF, SORT_IN_TEMPDB
= OFF )
```

While this command looks complicated, the bulk of the code is simply turning off various options. The `ALTER INDEX` command has a lot of options, some of which you can configure using the Maintenance Plan Wizard, but many more that you cannot. We will discuss all the available configuration options as we work through this chapter.

If you review the text file report from this task, it looks something similar to the following:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
Report was generated on "HAWAII."
Maintenance Plan: User Databases Maintenance Plan
Duration: 00:00:23
Status: Succeeded.
Details:
Rebuild Index (HAWAII)
Rebuild index on Local server connection
Databases: AdventureWorks
Object: Tables and views
Original amount of free space
Task start: 2009-07-29T16:01:48.
Task end: 2009-07-29T16:02:09.
Success
Command:USE [AdventureWorks]
GO
ALTER INDEX [PK_AWBuildVersion_SystemInformationID] ON [dbo].
[AWBuildVersion] REBUILD PARTITION = ALL WITH ( PAD_INDEX  = OFF,
STATISTICS_NORECOMPUTE  = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_
LOCKS  = ON, ONLINE = OFF, SORT_IN_TEMPDB = OFF )
GO
USE [AdventureWorks]
GO
ALTER INDEX [PK_DatabaseLog_DatabaseLogID] ON [dbo].[DatabaseLog]
REBUILD PARTITION = ALL WITH ( PAD_INDEX  = OFF, STATISTICS_
NORECOMPUTE  = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  =
ON, ONLINE = OFF, SORT_IN_TEMPDB = OFF )
GO
```

Retrieving Text file reports

*Unless you specified otherwise in the **Select Report Options** screen of the Wizard (see Chapter 3), text reports created by the Wizard are, by default, located in this folder: C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\JOBS.*

Once the header information is displayed, note that the command executes for each index in the database. I have abbreviated the report to only show two indexes being rebuilt, as there were a lot more that were actually rebuilt in the full report.

When and How Often to Rebuild Indexes

The performance of your indexes, and therefore your database queries, will degrade as you indexes become fragmented. The `Rebuild Index` task does a very good job of rebuilding indexes to remove logical fragmentation and empty space, and updating statistics. As such, it is very important that you schedule this task to run regularly.

On the other hand, the `Rebuild Index` task is resource intensive. In addition, as an index is being rebuilt, locks will be placed on it, preventing anyone from accessing it while the rebuilding occurs. Any queries trying to access this index in order to return the required results will be temporarily blocked, until the rebuild is complete. As such, the `Rebuild Index` task is considered an **offline activity**, to be run when as few people as possible are accessing a database. In general, this means during a scheduled maintenance window.

It is quite difficult to offer general advice with regard to when and how often to rebuild indexes using the Maintenance Plan Wizard, as it is so dependent on the nature of the data, the indexes and the queries that use them. However, take a look at my general advice with regard to index rebuilding, and then we'll consider the advice in a little more detail over the coming sections.

- **Nightly, if required**. If your indexes fragment rapidly, and you have a nightly maintenance window that allows you to run the `Rebuild Index` task, along with all the other maintenance tasks, then do so. Index fragmentation will degrade the performance of your indexes. Assuming that you have a maintenance window, rebuilding every night can't do any harm, and can very well boost the performance of your server.

- **Weekly, at minimum**. If you can't perform this task nightly, then, at a minimum, it should be run once a week, during a maintenance window. If you wait much longer than a week, you risk hurting your SQL Server's performance due to the negative impact of wasted empty space and logical fragmentation.

- **Consider alternatives, otherwise**. If you don't have a maintenance window long enough to run this task at least once a week, then you need to consider the following alternatives:

- **Use the online version of the** `Rebuild Index` **task** – available only with the Enterprise Edition of SQL Server.
- **Use the** `Reorganize Index` task **followed by the** `Update Statistics` task – if you're using the Standard Edition of SQL Server. This is your only real alternative when using the Maintenance Plan Wizard if want to avoid the `Rebuild Index` task.
- **Avoid the Maintenance Plan Wizard** – T-SQL or PowerShell scripts offer greater control and flexibility over the exact nature and duration of this task.

Tracking Index Fragmentation

The question of exactly *how often* to rebuild indexes is a difficult one to answer, and the Maintenance Plan Wizard doesn't offer any guidance. The speed and degree to which an index fragments depends on how it is used, and will vary wildly from database to database.

It is beyond the scope of this book to enter a full discussion of measuring index fragmentation, and therefore deciding how often you should rebuild your database's indexes. However, it is worth noting that the `sys.dm_db_index_physical_stats` Dynamic Management Function contains two columns that store valuable information regarding index fragmentation:

- `avg_page_space_used_in_percent` – this column stores the average amount of space that is used on a page. For example, a particular index might have 50% space used, which means that, on average, only half of the space on a data pageis used to store rows of data.
- `avg_fragmentation_in_percent` – this column stores the degree of logical fragmentation of an index, as a percentage. For example, a particular index might be 80% fragmented, which means that, on average, 80% of the data pages physical ordering does not match their logical ordering.

If you were to track this data over a period of time, you would be better able to gauge how quickly your indexes fragment, and so how often you should consider rebuilding them. However, if you are at this level, then the chances are high that you'll be using scripting techniques to rebuild your indexes rather than the Maintenance Plan Wizard.

Offline Index Maintenance

While it is not a requirement to perform the `Rebuild Index` task offline, while the database is not being accessed, it is certainly a strong recommendation, especially for large databases with many users. If your tables are relatively small, rebuilding will be fast and most users who happen to be accessing the database at the same time probably won't notice any performance

degradation as a result of the locking required by `Rebuild Index` task. On the other hand, if your tables are big, or if you have lots of concurrent users, the `Rebuild Index` task can negatively affect your users' experience, greatly slowing down their access to the database, and potentially causing some queries to time out from their client application.

Generally speaking, if you have a large enough maintenance window to accommodate running your `Rebuild Index` task offline, then I recommend you use this task, and run it during that window.

Online Index Maintenance

If you don't have a maintenance window, or it is not long enough to accommodate an offline `Rebuild Index` task, then you have one or two possible alternatives available to you, when using the Maintenance Plan Wizard:

- use the online version of the `Rebuild Index` task
- use the `Reorganize Index` task followed by the `Update Statistics` task.

If you have the Enterprise Edition of SQL Server, the Maintenance Plan Wizard offers a **Keep index online while reindexing** option, which means that the index will continue to be available to users even while it is being rebuilt. Even though this is an online activity, you will still want to schedule this task during a time of the day when the server is less busy, as it is still a resource-intensive activity. Performing this online task during busy times of the day can affect your users' ability to access the database in a timely manner, especially if your SQL Server already has performance bottlenecks.

If you don't have the Enterprise Edition, and your maintenance window is too short to accommodate an offline `Rebuild Index` task, then you should consider using the `Reorganize Index` task (see Chapter 8) instead, and running the `Update Statistics` task (see Chapter 9) immediately thereafter. The `Reorganize Index` task is an online operation, which means that it can run while users are accessing the database. While this is an online process, it is still resource intensive, and you should schedule the task during a time of the day when the server is less busy.

The downside to using the `Reorganize Index` task is that its index defragmentation capability is not as thorough and complete as the `Rebuild Index` task. In addition, it can take longer to run than the `Rebuild Index` task, and you have to run the `Update Statistics` task as a separate step.

Scripting Index Rebuilds

If you have the Enterprise Edition of SQL Server, chances are that your databases may be very large, and using the Maintenance Plan Wizard to maintain your databases may not be a great choice in the first place. You can obtain more flexibility and control by creating your own custom maintenance plans using T-SQL or PowerShell scripts.

For example, you can measure and track fragmentation using `sys.dm_db_index_physical_stats` and then build a script to defragment only those indexes that really need it.

Configuring the Rebuild Index Task

Now that we know a little about the `Rebuild Index` task, and when it should be run, let's look at its configuration screen from the Maintenance Plan Wizard, shown in Figure 7.1.

Our first choice is to select which databases we want to run this task against. We have already covered the various options in detail, in Chapter 3, so here we'll just focus on specific considerations for this task.

Database Selection

First, notice the **Databases** drop-down box appears on the screen as we have seen before. Second, notice that directly below the **Databases** drop-down box are two more drop-down boxes we have not seen before: **Object** and **Selection**. These two drop-down boxes appear for some tasks, and not others. We will see what they do in a moment; I just wanted to point them out now so you will be ready when I begin talking about them.

Selecting Several Databases

As a general rule, you want to keep the number of separate Maintenance Plans to a minimum so, ideally, you'd create a single plan and apply the `Rebuild Index` task to all indexes in a given set of databases, for example in all **user** databases. Also, in order to ease maintenance and avoid confusion, each task in the plan should be applied to the same set of databases.

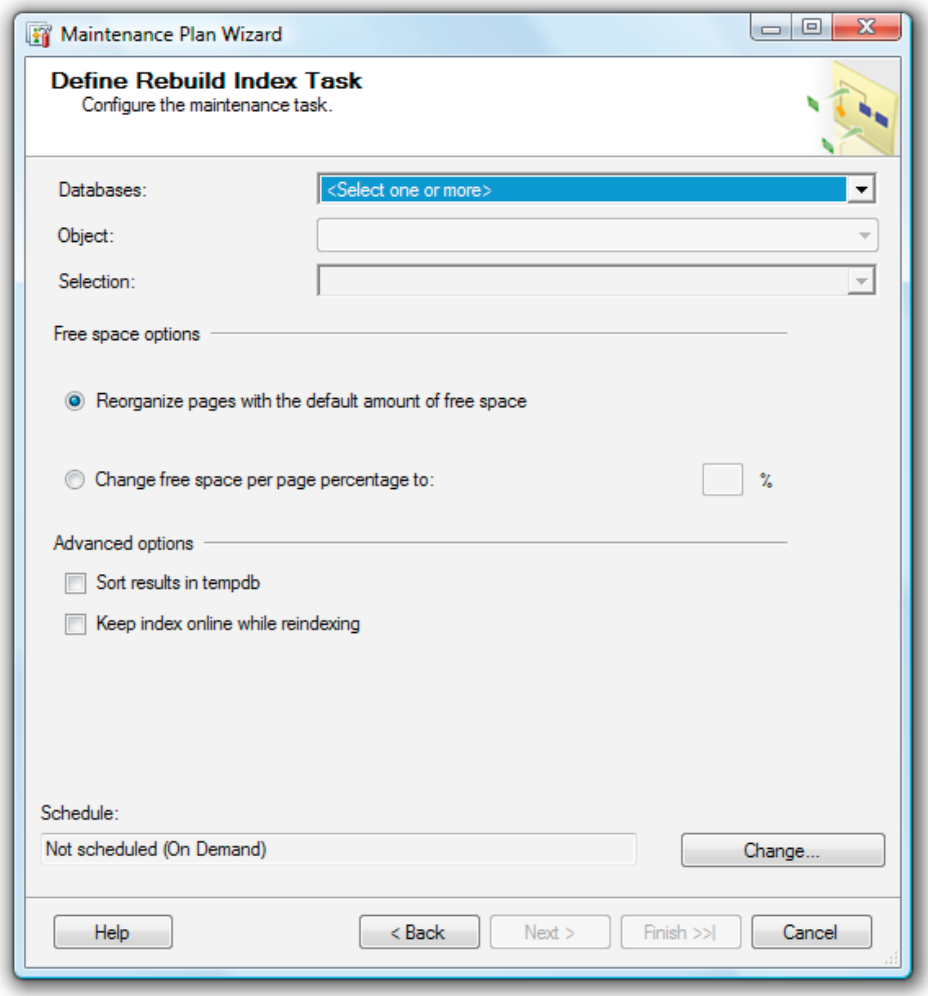

Figure 7.1: We are ready to configure the "Rebuild Index" task.

However, there may be special cases where you'd need to create separate plans to deal with the specific index maintenance requirements of different databases. For example, let's assume that on a single SQL Server instance you have 25 small databases, each less than 1 GB in size, and one large database, say, 50 GB. Let's also assume that few, if any, users will need access to the small databases during your maintenance windows, but that many users may need to access the 50 GB database during this time. In this case, you might consider creating a special Maintenance Plan for the 50 GB database that uses the `Reorganize Index` and `Update Statistics` tasks, and another Maintenance Plan that applies the `Rebuild Index` task to the smaller databases.

For this example, we are going to keep things simple, so let's assume we want to perform the `Rebuild Index` task for all user databases. In this case, choose the option shown in Figure 7.2, and then click **OK**.

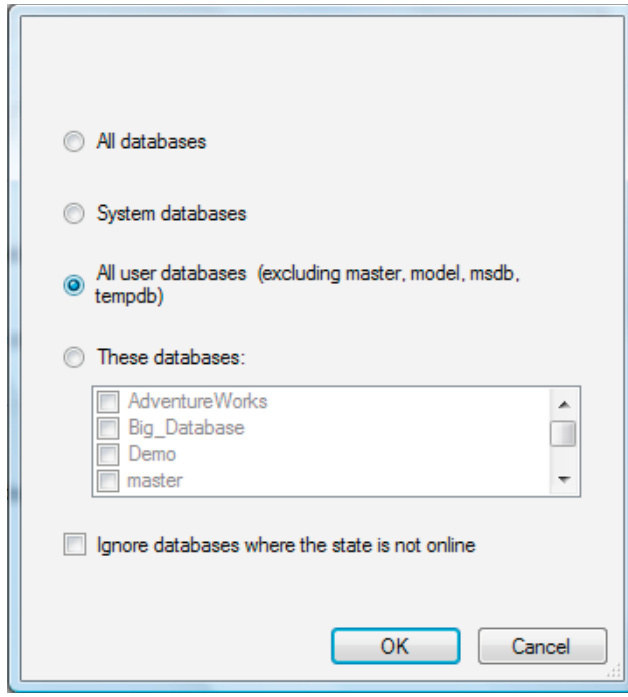

Figure 7.2: To keep things simple, select the "All user databases" option.

The **Define Rebuild Index Task** screen reappears, and the two drop-down boxes I referred to earlier are displayed below the Databases drop-down box, but they are grayed out, as shown in Figure 7.3.

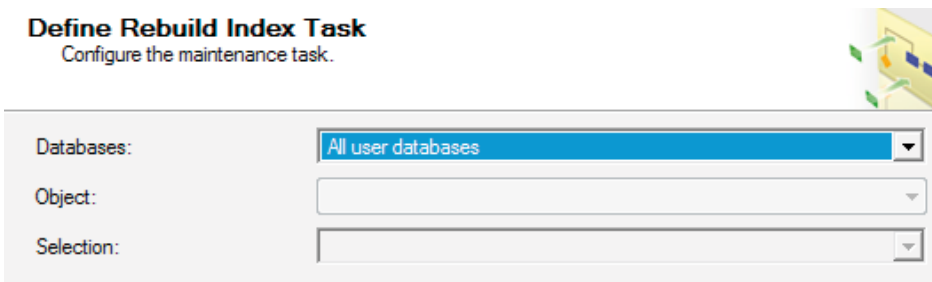

Figure 7.3: The "Object" and "Selection" drop down boxes are not available.

So what's going on? Why are these two options grayed out? The reason is that these two options are only available if you select **one** database on which to run the `Rebuild Index` task. Since we selected **All user databases**, they are not available.

Selecting a Specific Database

Although it does not apply to our example, let's take a look at what happens if you select only a single database, such as `AdventureWorks`, for the task. To do this, select **These databases** from the screen shown in Figure 7.2 and then check the checkbox for **AdventureWorks**. When you click **OK**, this section of the **Define Rebuild Index Task** screen will look as shown in Figure 7.4.

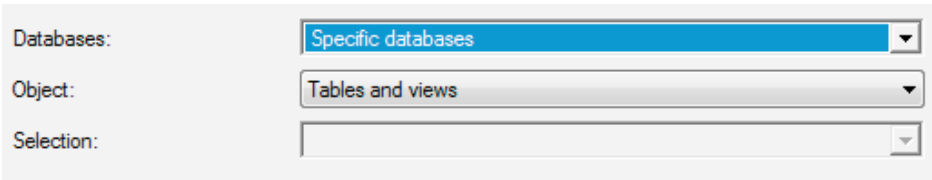

Figure 7.4: When a single database is selected, then the "Object" drop-down box becomes available.

Notice that **Specific databases** now appears in the **Databases** drop-down box, the **Object** box is now available, and the **Selection** box is still, for the time being, grayed out.

What the **Object** and **Selection** options allow you to do is to selectively rebuild some of the indexes in your database, and not others. If you click on the **Object** drop-down box, you'll see the choices as shown in Figure 7.5.

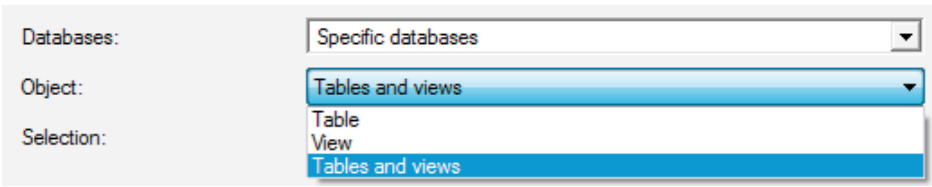

Figure 7.5: You must select either "Table" or "View."

Notice that there are three choices for **Object**. If you leave the default option selected, **Tables and views**, then the `Rebuild Index` task will be applied to the indexes associated with all tables and all indexed views in the selected database. In other words, you haven't changed anything. In order to narrow the scope of the task to specific objects, you need to choose either **Table** or **View**. Having done this, the **Selection** drop-down box becomes available. For example, choose **Table**, and then click on "Select one or more" in the now available **Selection** drop-down box, as shown in Figure 7.6.

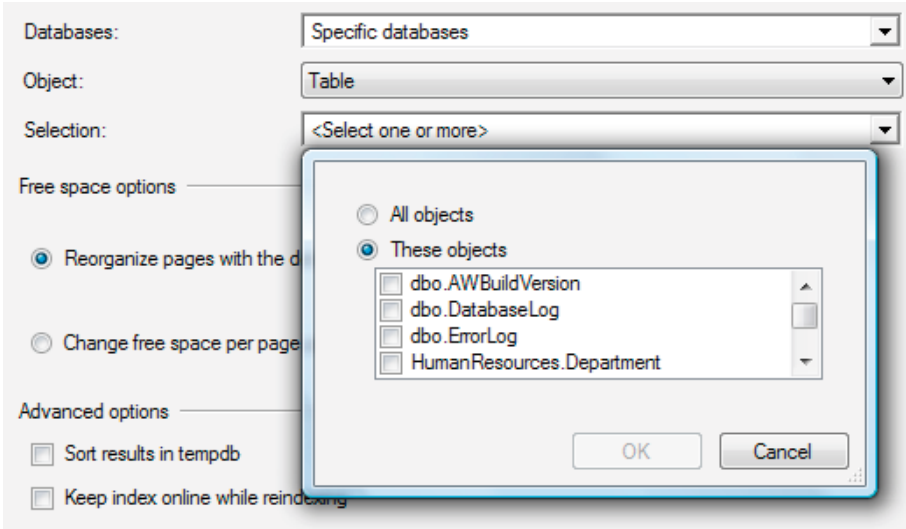

Figure 7.6: You can select which tables you want to rebuild with the Rebuild Index task.

Now, you get the option of selecting specific tables within the `AdventureWorks` database, to which this task should apply. For example, you could choose to rebuild only the indexes associated with the `dbo.ErrorLog` table, or you could select some combination of tables, by checking each of the relevant checkboxes.

Why would you want to rebuild the indexes for some tables and not others? Actually, there is a very good reason for this. In most databases, there are some tables that are virtually static; they rarely if ever change, and so there is no benefit in rebuilding their associated indexes as they don't, over time, develop wasted empty space or become logically fragmented. By selecting only those indexes that really need defragmenting, you can reduce the time it takes to perform the `Rebuild Index` task and, at the same time, reduce the resource overhead associated with this task.

The problem I see is that most people who are using the Maintenance Wizard won't have the knowledge to determine which indexes are relatively static and which are subject to a lot of wasted space and logical fragmentation. If you are at the level where you know how to evaluate each index using the `sys.dm_db_index_physical_stats` DMF, in order to apply a selective rebuild process, then the chances are you are probably better off implementing this process using T-SQL or PowerShell scripts, and avoiding use of the Maintenance Plan Wizard in the first place.

Before we move on, let's briefly consider the **View** option that is available in the **Object** drop-down box, as shown in Figure 7.7.

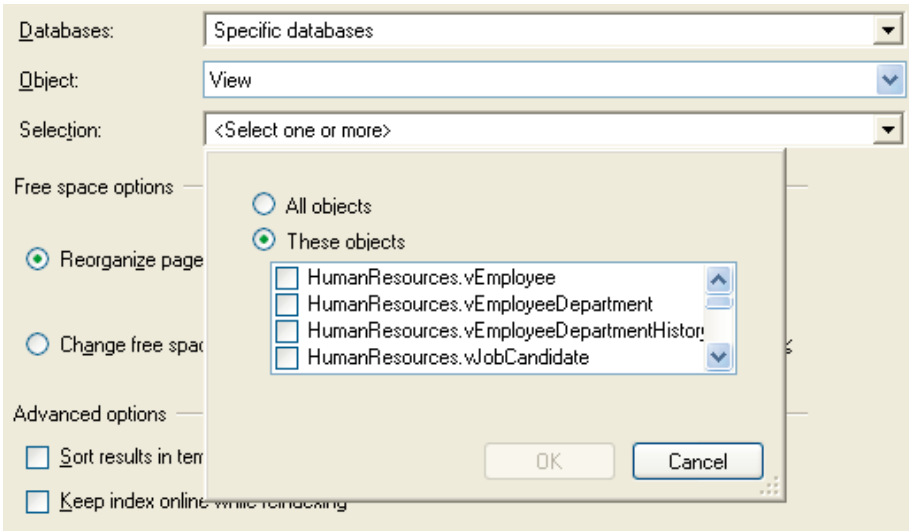

Figure 7.7: You can select which indexed views you want to rebuild with the Rebuild Index task.

In this case, **View** doesn't refer to conventional views, but to indexed views. Indexed views are physical views, unlike regular views, which are only materialized when they are called by a query. Because indexed views are physical, they need rebuilding just like regular indexes. As per my advice with regard to the **Table** option, if you need this kind of granularity for the maintenance of your indexes, you shouldn't be using the Maintenance Plan Wizard for this task.

While I have taken a little time to explain what the Object and Selection drop-down boxes do, I am recommending that you don't use then, as they just make Maintenance Plans overly complicated, defeating the benefit of using them in the first place.

Free space options

We still have several more choices to make before we are done configuring this task. Note that the discussion of these options assumes that each of your tables has a clustered index, and is not a heap. A heap is a table without a clustered index. As a best practice, all tables should have a clustered index.

The first two choices are listed under **Free space options** and include **Reorganize pages with the default amount of free space** and **Change free space per page percentage to**, as shown in Figure 7.8. You can choose one option or the other, but not both.

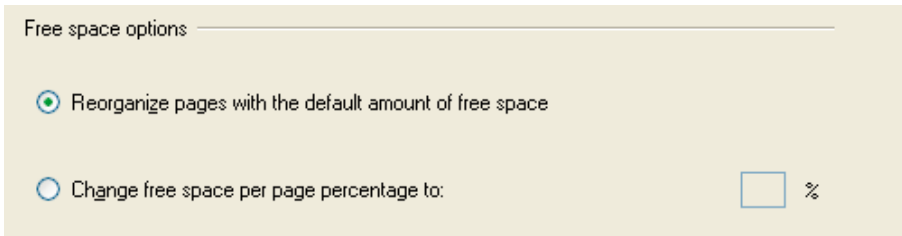

Figure 7.8: These options can have a significant impact on the Rebuild Index task.

The default option of **Reorganize pages with the default amount of free space** is a little confusing. First, it says *reorganize*, not *rebuild*. Remember, we are working on the `Rebuild Index` task, not the `Reorganize Index` task. Don't let this confuse you into thinking that selecting this option reorganizes indexes, rather than rebuilding them. It does the latter, and this is actually a mistake in the user interface. It really should say "rebuild," not "reorganize."

The second part of this first option says "default amount of free space." What does that mean? When creating a SQL Server index, there is an option to create the index with a certain amount of free space on each data page. This setting is known as the **fill factor**. If an index is created without specifying a fill factor, then the default fill factor is used, which is 100 (actually 0, but 0 means the same thing as a 100% fill factor). This means that no free space is created for the data pages of an index.

The potential problem with a fill factor of 100 arises when data is added to a table as a result of an `INSERT` or `UPDATE`, and a new row needs to be added to a data page. If there is no room for it, then SQL Server will reorganize the rows, moving some of the rows onto a new data page, and leaving some on the old data page. This is known as **page splitting**. While page splitting is a normal SQL Server activity, too much page splitting can cause performance

issues because it results in index fragmentation, the very thing we are trying to eliminate with the `Rebuild Index` task. In order to mitigate this problem, DBAs often decrease the fill factor to perhaps 90, meaning that data pages will be 90% full, leaving 10% free space.

> ***For more information regarding fill factors and page splitting...***
>
> *...refer to Books Online. A full discussion of these topics is beyond the scope of this book, but I needed to include a little background so you could better understand what is happening when you make particular selections within the Wizard. Also, don't assume that my example of a fill factor of 90 is appropriate for your indexes. It may be, or it may not be.*

What is really confusing is that the phrase "default amount of free space" in the Wizard does not mean the same thing as the "default fill factor" that can be set for the entire server. Some people confuse the two.

In the `Rebuild Index` task, "default amount of free space" refers to the fill factor that was used when a specific index was first built, or last rebuilt. In other words, if you choose the option **Reorganize pages with the default amount of free space**, what happens is that each index is rebuilt using whatever fill factor value was used the last time it was rebuilt. This may be the same as the server-wide default, or it may be a specific value that was specified for that index, or it may be a value set using the second **Change free space per page percentage to** option (discussed next).

In almost all cases the "default amount of free space" option is the one you want to use, as it means the index will be rebuilt using the fill factor that was originally specified when the index was created.

With the second option, **Change free space per page percentage to**, you specify a single fill factor value to be used for every index when it is rebuilt. For example, if you choose **Change free space per page percentage to** and set it to 10%, this is the same thing as setting all of the indexes in your database to a fill factor of 90, regardless of what the value was when the index was created. It is rarely a good idea for every index in your database to have the same fill factor. The appropriate fill factor is specific to an index, and you can't generalize a fill factor that will work well for every index in your database. While this setting might be beneficial for some indexes, it could cause performance problems with others. As a result, I advise against using this option.

Of course, the choice of the default **Reorganize pages with the default amount of free space** option assumes that the fill factors of all of your indexes have been ideally set when they were originally created, or were last rebuilt. If they aren't, then it's a tossup as to which option is really the best. But, assuming that you don't know if the fill factors are ideal or not, which you probably don't, I would still recommend using this default option.

Advanced options

The two options under **Advanced options** are shown in Figure 7.9.

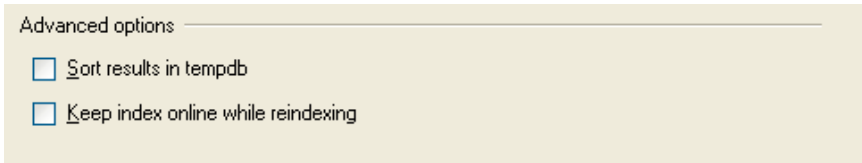

Figure 7.9: The Advanced options section of the Define Rebuild Index Task screen

By default, both options are turned off. The first one is **Sort results in tempdb**. If you **don't** choose this option then, when an index is rebuilt, all of the rebuilding activity is performed in the database file itself. If you select the **Sort results in tempdb** option, then some of the activity is still performed in the database, but some of it is also performed in `tempdb`. The benefit is that this can often speed up the index rebuild process. The drawback is that it also takes up a little more overall disk space, as space in `tempdb` is required, in addition to some space in the database where the indexes are being rebuilt.

The benefit you get out of this option depends on where `tempdb` is located on your server. If `tempdb` is located on the same drive or array as the database file that is having its indexes rebuilt, then the benefit may be minimal, if any. However, if `tempdb` is located on its own isolated drive spindles, then the benefit will be greater because there is less disk I/O contention.

So, should you use this option? If your databases are small, you probably won't be able to discern much performance benefit, but if you have large databases, with large tables and indexes, and if `tempdb` is located on its own spindles, then turning this feature on will probably boost index rebuild performance.

The second advanced option is one we've discussed previously: **Keep index online while reindexing**. This option is only available if you have the Enterprise Edition of SQL Server. By selecting this option, index rebuilding becomes an online, rather than offline task. If you are using Enterprise Edition, you will probably want to select this option. I say "probably" because there are pros and cons to performing an online index rebuild – a topic that is beyond the scope of this book.

Creating the Job Schedule

As always, our final step is to define an appropriate schedule on which to run our `Rebuild Index` job. With the previous advice in mind, the best option would be to run the job within a nightly maintenance window. However, in Chapter 3, I stated that my assumption for all my examples was that I have a single weekly maintenance window, which is the entire day of Sunday.

Therefore, let's schedule the `Rebuild Index` task to occur on Sunday, right after the `Check Database Integrity` task completes. As such, the screen will look as shown in Figure 7.10.

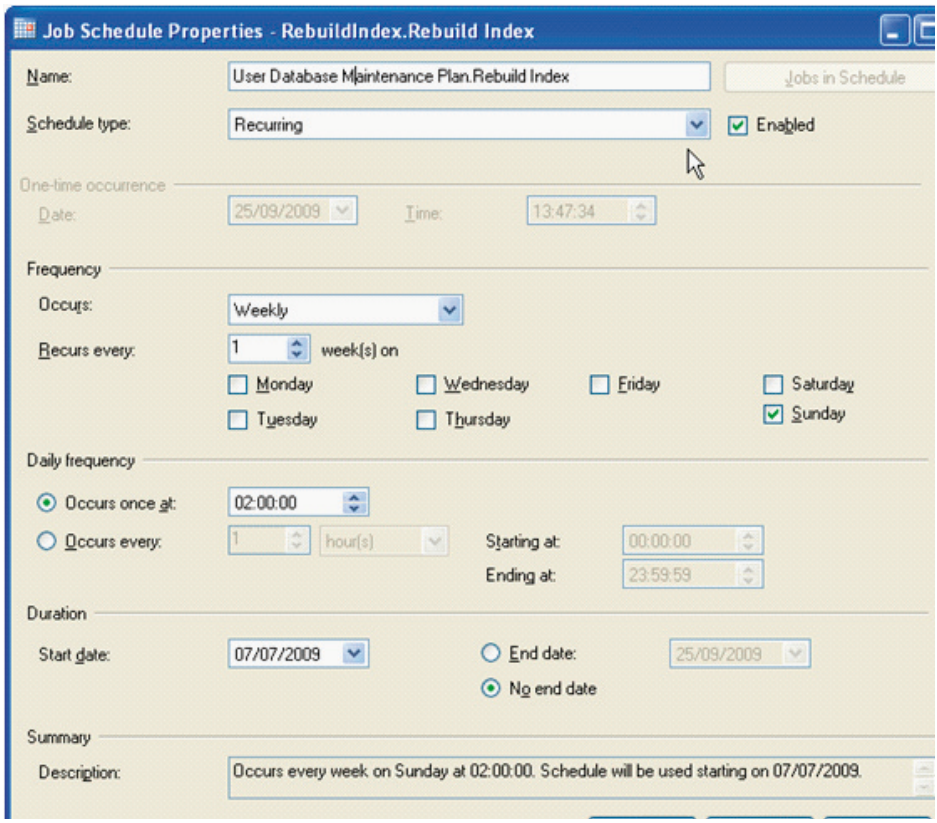

Figure 7.10: The schedule for the Rebuild Index job.

The only question you need to consider is how soon after running the `Check Database Integrity` task should you schedule a `Database Rebuild` task. That depends on how long the `Check Database Integrity` task takes to complete, and you won't know until you try it.

Since this is a new Maintenance Plan, we don't have any experience with regard to how long each task runs yet, and so we have to guess. In this example, I'll guess that the first `Check Database Integrity` task will take an hour, starting at 1 a.m., so I will schedule the `Rebuild Index` task to start at 2 a.m. If I'm wrong, the two jobs will overlap, which could cause some performance problems.

As a DBA, the first time you run any Maintenance Plan, you need to check how long each job takes to run, as described in the "Avoid Overlapping Tasks" section of Chapter 4. If your guess is wrong, and jobs overlap, you can use the Maintenance Plan Designer (see Chapters 16–19) to alter the schedule for the next time it runs.

I recommend that you run the `Rebuild Index` task before any of the backup tasks (discussed in Chapters 12, 13 and 14) are performed. This way, if you have to restore a backup, your backup will be of the latest, index rebuilt version.

Summary

Index fragmentation is an issue all databases experience and, if it is not removed on a regular basis, it can lead to query performance problems. One way to remove index fragmentation is to regularly run the `Rebuild Index` task, which drops and rebuilds every index in a database. While the `Rebuild Index` task is very effective at what it does, it is considered an offline activity, and it is very resource intensive. As such, using this task may not always be appropriate for all your databases. In the next chapter, we take a look at an alternative to the `Rebuild Index` task, that is, the `Reorganize Index` task.

Chapter 8: Reorganize Index Task

In some situations, the `Reorganize Index` task, coupled with the `Update Statistics` task (covered in the next chapter), provides a useful alternative to the `Rebuild Index` task. But before I go any further, let me state again, very clearly, that you should **not** run the `Reorganize Index` task if you are running the `Rebuild Index` task. These tasks perform essentially the same function, and there is no point in repeating the same step. Unfortunately, the Maintenance Plan Wizard is not smart enough to prevent you from making this mistake, and will let you create and schedule both of them.

Since `Reorganize Index` works to achieve essentially the same goal as the `Rebuild Index` task, namely maintaining the health of your database indexes, many of the arguments regarding when to use the task, how often, and so on, are exactly the same as those discussed in the previous chapter. Therefore, I will not repeat them here.

An Overview of the Reorganize Index Task

Like the `Rebuild Index` task, the `Reorganize Index` task works to minimize wasted space and logical fragmentation in database indexes. It can be regarded in some ways as a lightweight alternative to the `Rebuild Index` task. As the name suggests, `Rebuild Index` drops the index and rebuilds it from scratch. `Reorganize Index` is more of a gentle reshuffling of the leaf-level pages of an index, such that the physical ordering matches the logical ordering, and wasted space is minimized.

Because `Reorganize Index` is an online operation, queries can still access the indexes even while they are being reorganized, though some deterioration in performance may be noticed. In addition, this task does not automatically update statistics like the `Rebuild Index` task does, so this action must be performed separately, using the `Update Statistics` task.

Using its default settings, the `Reorganize Index` task runs the following T-SQL statement for every table in the selected databases.

```
ALTER INDEX index_name ON table_name REORGANIZE WITH ( LOB_
COMPACTION = ON )
```

Notice that this command uses the `ALTER INDEX` command with a single option, which we will discuss shortly.

After the `Reorganize Task` runs, it produces a text report similar to the following:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: MaintenancePlan
Duration: 00:00:15
Status: Succeeded.
Details:
Reorganize Index (HAWAII)
Reorganize index on Local server connection
Databases: AdventureWorks
Object: Tables and views
Compact large objects
Task start: 2009-07-30T15:05:51.
Task end: 2009-07-30T15:06:06.
Success
Command:USE [AdventureWorks]
GO
ALTER INDEX [PK_AWBuildVersion_SystemInformationID] ON [dbo].
[AWBuildVersion] REORGANIZE WITH ( LOB_COMPACTION = ON )
GO
USE [AdventureWorks]
GO
ALTER INDEX [PK_DatabaseLog_DatabaseLogID] ON [dbo].[DatabaseLog]
REORGANIZE WITH ( LOB_COMPACTION = ON )
GO
```

While the above is an abbreviated report, yours will show the `ALTER INDEX` command run for every index in every table in your selected databases. If there are any problems or error messages, you will see them here also.

Reorganize Versus Rebuild

Now that you have a little background about Rebuild and Reorganize, I think it's time for a more comprehensive summary of the pros and cons of using the `Reorganize Index` task versus using the `Rebuild Index` task. With this information, you will be better able to make an educated decision on which option is right for your particular circumstances.

	Reorganize Index Task	Rebuild Index Task
Removing empty space and logical fragmentation	Performs a less thorough index defragmentation than Rebuild Index. If an index does not have any fragmentation, then it is not reorganized, saving resources.	Virtually all wasted free space and logical fragmentation is removed. All indexes are rebuilt from scratch, whether they need it or not.
Performance impact	Does not require long blocking locks. An online task that allows users to access the database during the task.	Requires potentially long blocking locks that prevent users from accessing the indexes being rebuilt. A task that should be performed offline, though with the Enterprise Edition of SQL Server, you can use the online version of rebuilding an index.
Speed	Generally takes longer to run than the Rebuild Index Task.	Generally runs faster than the Reorganize Index Task.
Space requirements	Uses less disk space than the Rebuild Index Task. Uses less space in the transaction log than the Rebuild Index Task.	Uses more disk space than the Reorganize Index Task. More space is required in the transaction log than the Reorganize Index task.
Statistics maintenance	Index and column statistics must be updated separately. This adds to the administration hassle.	Index and column statistics are automatically updated as part of this step, using the FULLSCAN option.

When and How Often to Reorganize Indexes

As I stated in the previous chapter, my general preference is to use the `Rebuild Index` task, as long as it fits into my available maintenance window. If I don't have an available maintenance window, then I generally use the `Reorganize Index` task, along with the `Update Statistics` task. So, before you choose Reorganize over Rebuild, or vice versa, you need to determine what your maintenance windows are.

While the `Reorganize Index` task does offer you the option to run the task outside maintenance windows, I would still advise against this if possible. Although the performance impact of this task, along with `Update Statistics`, will be much lower than for the `Rebuild Index` task, it could still be felt by the users, especially if your database tables are large and your servers busy.

As such, if you've chosen the `Reorganize Index` task, here are my general scheduling recommendations, when using the Maintenance Plan Wizard.

- **Nightly, if possible**. If running both the `Reorganize Index` task and `Update Statistics` task does not significantly affect users, I suggest you run these two tasks daily, picking a time when running them will have the least impact.

- **Consider alternatives, otherwise**. If running the `Reorganize Index` task and `Update Statistics` task does affect user performance, then you may need to consider exerting a finer-grained control over the process, using T-SQL or PowerShell scripts.

Configuring the Reorganize Index Task

Now that we have a basic understanding of when you might use the `Reorganize Index` Task, let's take a look at how to configure it, using the Maintenance Plan Wizard. The **Define Reorganize Index Task** screen is similar to the **Define Rebuild Index Task** screen, but it has fewer options, as shown in Figure 8.1.

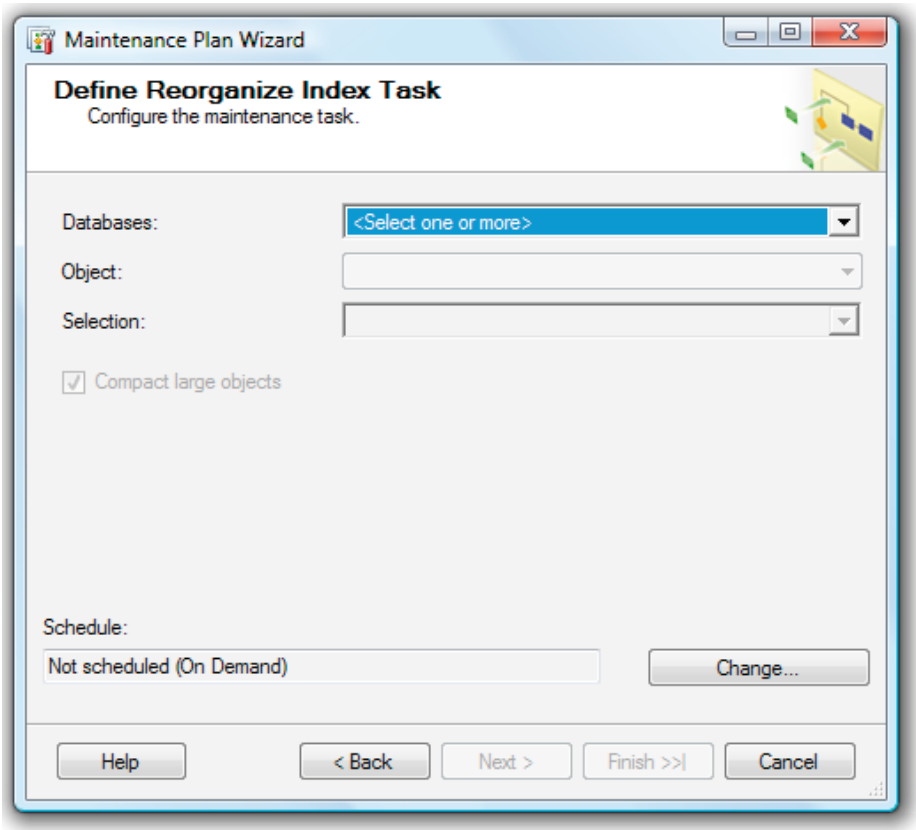

Figure 8.1: The Reorganize Index Task screen is similar to the Rebuild Index Task, but it has fewer options.

Database Selection

The options here, with the **Databases**, **Object** and **Selection** drop-down boxes, are the same as those described in the equivalent section of the Rebuild Index chapter (Chapter 7), so I will not walk through them again.

As before, my advice is that you select the same database or databases here as for every other task that comprises the plan. There may be some special cases where you need to use one plan for one set of databases, using the `Reorganize Index` task, and a different plan for other databases, using the `Rebuild Index` task. However, this does begin to negate one of the big selling points of using the Maintenance Plan Wizard: **simplicity**. If you need to use both the `Reorganize Index` task and the `Rebuild Index` task on the same SQL Server instance, you may be better off creating a manual maintenance plan using T-SQL or PowerShell.

Don't be tempted to configure Rebuild and Reorganize in a single plan

While it is possible to create a single Maintenance Plan that will run the `Reorganize Index` *task and the* `Update Statistics` *task for some databases, and to run the* `Rebuild Index` *task for other databases, this can get confusing very quickly. Instead, create separate Maintenance Plans, as I described previously.*

As with the `Rebuild Index` task, you also have the ability to select an individual database, and then narrow the scope of the task to one or more tables or indexed views. Just as I suggested you should not use this feature with the `Rebuild Index` task, I make the same recommendation here. If you are at the point where you need to pick and choose which tables and indexed views to reorganize, then you would be better off using T-SQL or PowerShell scripts to do this task for you, as the Maintenance Plan Wizard is very inflexible.

Compact large objects

The only major task-specific option that we need to consider is whether or not to select the **Compact large objects** option, as shown in Figure 8.2.

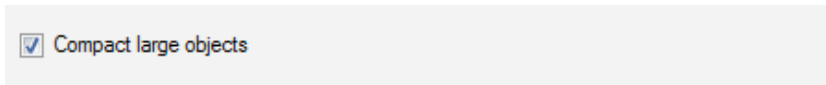

Figure 8.2: Generally, you will want to keep this option selected.

It is checked by default, which means that if a table on which this task runs stores `LOB` data (`text`, `ntext`, or `image` data types), then `LOB` data in these tables will be treated just the same as any other type of data, and will be reorganized.

If you deselect this option, then `LOB` data will not be reorganized. Generally, you will want to keep this option selected, as reorganizing `LOB` data can boost the performance of your database. Of course, if you have a lot of `LOB` data, then this task will take more time, which is to be expected.

If you don't want to take this extra time, or you don't care about `LOB` data compaction, then you can turn this option off. When you do, the `ALTER INDEX` command will change to:

```
ALTER INDEX index_name ON table_name REORGANIZE WITH ( LOB_
COMPACTION = OFF )
```

Creating the Job Schedule

The last option on the Wizard screen is **Schedule** which we already know how to use. In our example, we'd run the `Reorganize Index` task as a direct replacement of the `Rebuild Index` task, in other words, at 2 a.m. on Sunday morning, during our maintenance window (see Figure 7.10).

In general, I recommend you run the `Reorganize Index` task after the `Check Database Integrity` task, but before any of the backup tasks. This way you won't waste any time reorganizing your database should the `Check Database Integrity` task fail. In addition, your backups will be of the latest, reorganized version so, if you have to restore it, it will be ready to use.

Once the `Reorganize Index` task has run, then you want to immediately start the `Update Statistics` task, so that the databases have properly updated statistics. Don't accidently overlap these tasks, as they will both fight for resources, potentially contributing to performance problems on your SQL Server instance.

Summary

Like the `Rebuild Index` task, the `Reorganize Index` task works to minimize wasted space and logical fragmentation in database indexes. This can help boost the performance of your SQL Server. Think of it as a lightweight alternative to the `Rebuild Index` task, which is best used when you don't have an available maintenance window to perform the `Rebuild Index` task.

In the next chapter, we take an in-depth look at the `Update Statistics` task, which is a task you will always want to run after using the `Reorganize Index` task.

Chapter 9: Update Statistics Task

The `Update Statistics` task has been referenced many times in previous chapters, and now it's time to investigate exactly what it does and how it works. When the `Update Statistics` task is run, it executes the `UPDATE STATISTICS` command against all of the tables in the databases you select, bringing up to date all index and column statistics.

As discussed in Chapter 7, the `Rebuild Index` task automatically updates statistics and so you should not run the `Update Statistics` task after running the `Rebuild Index` task. To do so would, at best, be a waste of server resources and could, if you choose the incorrect configuration options for the `Update Statistics` task, actually *degrade* the quality of the statistics available to the query optimizer.

Conversely, the `Reorganize Index` task, discussed in Chapter, does not update statistics and so should immediately be followed by execution of the `Update Statistics` task.

Overview of the Update Statistics Task

When a query is submitted to SQL Server, the Query Optimizer attempts to work out the best way to execute that query. It will generate a series of possible execution plans for the query, and assess the predicted overall cost of each plan, in terms of CPU time, I/O, execution time, and so on. The plan with the lowest estimated cost is used to execute the query. This description is an oversimplification of how the Query Optimizer works, but it is adequate for the purposes of this discussion.

In order to accurately assess the relative costs of each potential execution plan, the optimizer relies on column and index **statistics** that are maintained by the SQL Server engine.

SQL Server examines the rows of data in the database (or a percentage of those rows – see later) and generates and maintains Statistics objects that provide the Query Optimizer with information such as the size, the number and structure of the tables, the distribution of data values within the table columns, the number of rows that will be returned, the number and structure of available indexes, and index selectivity. Based on these statistics, it decides whether or not indexes can be used, the cost of various types of joins, and so on, and arrives at what it believes is the optimal execution plan.

If the statistics available to the query optimizer are out of date or incomplete, then it might create a suboptimal query execution plan, resulting in a poorly performing query.

To some extent these statistics are self-maintaining. Column and index statistics are automatically created, and regularly updated, as long as the following two options are turned on for a given database (which they are, by default):

- `AUTO_CREATE_STATISTICS` – used by the optimizer to create statistics on individual columns in the query predicate, as required

- `AUTO_UPDATE_STATISTICS` – the query optimizer automatically updates index and column statistics if it determines that they may be out of date, such as when a data `INSERT`, `UPDATE` or `DELETE` operation changes the data distribution.

In most cases, SQL Server does a fairly good job at keeping statistics up to date. However, if you have just performed a database maintenance task, such as reorganizing indexes, then you need to manually update the statistics to ensure that the optimizer has the accurate information it needs to optimize query execution plans. Also, if your SQL Server instance is suffering from poor or erratic performance for certain queries, then you may need to consider manually updating statistics. This is where the `Update Statistics` task comes into play, although in either case it is generally recommended that you perform these manual updates in addition to leaving `AUTO_UPDATE_STATISTICS` turned on.

> ***Manually Creating Statistics***
>
> *In addition to manually updating statistics, there may also be occasions when you need to create more detailed column statistics than are provided by* `AUTO_CREATE_STATISTICS`*. You can do this using the* `CREATE STATISTICS` *command. This task is not covered by the Maintenance Plan Wizard and is outside the scope of this book. More information can be found in Books Online.*

When the `Update Statistics` task runs using its default settings, the following T-SQL code is executed on every table in every selected database.

```
UPDATE STATISTICS table_name WITH FULLSCAN
```

We will discuss the `FULLSCAN` option a little later in this chapter, but basically it means that the optimizer will check every row of every table in order to ensure that the index and column statistics are as accurate as possible.

When you execute this task, it produces a text report similar to this:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: MaintenancePlan
Duration: 00:00:07
Status: Succeeded.
Details:
Update Statistics (HAWAII)
Update Statistics on Local server connection
Databases: AdventureWorks
Object: Tables and views
All existing statistics
Task start: 2009-07-30T15:25:13.
Task end: 2009-07-30T15:25:19.
Success
Command:use [AdventureWorks]
GO
UPDATE STATISTICS [dbo].[AWBuildVersion]
WITH FULLSCAN
GO
use [AdventureWorks]|
GO
UPDATE STATISTICS [dbo].[DatabaseLog]
WITH FULLSCAN
GO
```

While the above is an abbreviated report, yours will show the `UPDATE STATISTICS` command run for every table in your selected databases. If there are any problems or error messages, you will see them here also.

When and How Often to Update Statistics

Running the `Update Statistics` task is an online procedure and generally doesn't have much negative impact on a server's performance, especially for smaller servers with low numbers of users. However, bear in mind that your cached query plans reference your existing `Statistics` objects. What this means is that, when you update those `Statistics` objects, the plans that reference them will need to be recompiled, which could be an expensive operations terms of server resources. If you're updating the statistics too

frequently, you can cause unnecessary recompiles and negatively affect performance, especially if your databases are large and user numbers high.

> ***Statistics sampling***
>
> *If your databases are large, you may consider configuring the task to sample only a percentage of the rows in the database, in order to reduce the burden on the server. The downside is that this reduces the accuracy of the statistics. This is discussed in more detail shortly, in the section on the Scan type option.*

With this in mind, here is my general advice with regard to when and how often to run the `Update Statistics` task.

- **Never**, if you are running frequent (e.g. nightly) index rebuilds. The `Rebuild Index` task automatically performs a full scan statistics update of all indexes and columns.
- **Immediately after** the `Index Reorganization` task. So if you run the `Index Reorganize` task in a nightly maintenance window, you will also run a nightly `Update Statistics` task.
- On days when you don't run the `Rebuild Index` or the `Reorganize Index` task. See why below.

Here's something to consider. Let's say that your maintenance window only allows you to perform a weekly `Rebuild Index` task, or a `Reorganize Index` task followed by an `Update Statistics` task. When using this particular scheduling, you may discover that query performance among some of your databases is uneven. In other words, sometimes a particular query runs very fast, and other times it runs very slowly. While there are many possible causes for this, the problem may be caused by, or exacerbated by, incomplete or out of date statistics.

Assuming that you have determined that outdated or incomplete statistics are causing the erratic performance behavior of some queries, one way to help prevent this problem is to run the `Update Statistics` task on those nights when you are not running the `Rebuild Index` task or the `Reorganize Index` task. Doing so can help to ensure that your databases' index and columns statistics are up to date, helping to optimize query performance.

Configuring the Update Statistics Task

Now that we know what the `Update Statistics` task does, let's learn how to configure it. The **Define Update Statistics Task** screen in the Maintenance Plan Wizard is shown in Figure 9.1.

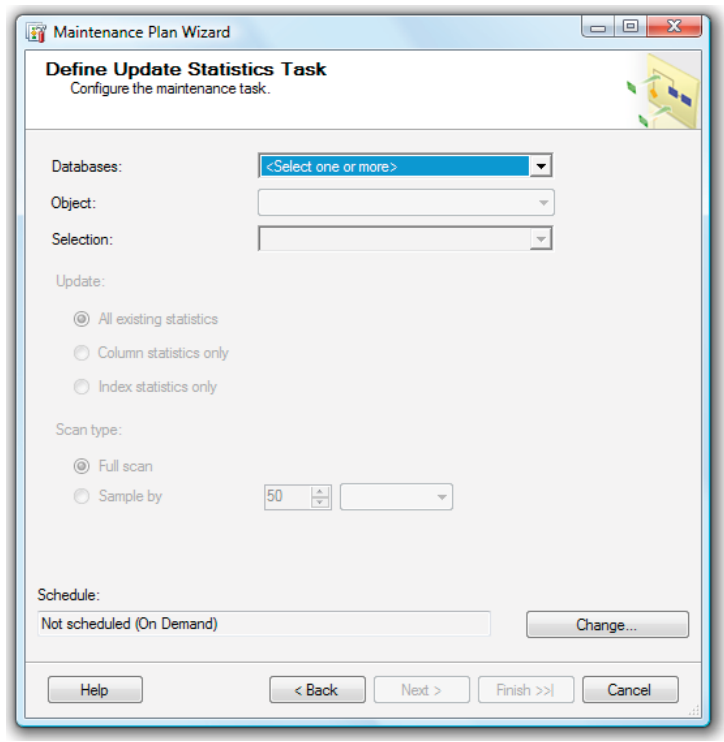

Figure 9.1: The Update Statistics Task screen has both familiar and unfamiliar configuration options.

The first part of the **Define Update Statistics Task** screen looks similar to the **Rebuild Index Task** and the **Reorganize Index Task** screens.

Database Selection

The options here, with the **Databases**, **Object** and **Selection** drop-down boxes, are the same as those described in the equivalent section of **Rebuild Index** chapter (Chapter 7), so I will not explain them again here. On the whole, I recommend you don't use the **Object** and **Selection** options for this task. If you need this kind of granularity in your Maintenance Plan, then you should be using T-SQL or PowerShell scripts instead.

If you are using the `Reorganize Index` task as part of your Maintenance Plan, then you should select the same databases here as you selected for the `Reorganize Index` task. As discussed previously, any database that is reorganized needs to have its statistics updated.

If you are creating a special Maintenance Plan that will only run the `Update Statistics` task (for example, on the days that you aren't running the `Reorganize Index` task or the `Rebuild Index` task), then you will most likely want to select all your user databases.

Once you have selected your databases, the **Update** and **Scan type** options become available, as shown in Figure 9.2.

Let's take a look at each option in turn.

The Update Option

The **Update** option allows you to specify which types of statistics are to be updated. The options are fairly self-explanatory.

- **All existing statistics** – both column and index statistics are updated.
- **Column statistics only** – only updates column statistics.
- **Index statistics only** – only updates index statistics.

The default is **All existing statistics** and is the correct choice in almost all cases, as both types of statistics need to be updated if you want the query optimizer to have all the data it needs to create optimal query plans.

The only reason you might want to choose either of the other options is if you want to reduce the amount of time this job takes to execute. However, by doing this, you increase the risk that your query execution plans may be less than ideal.

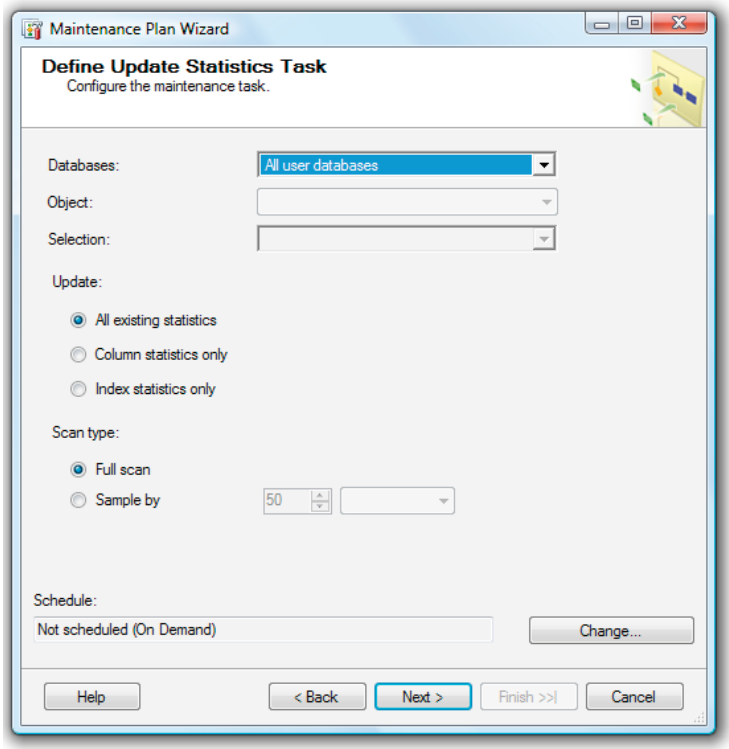

Figure 9.2: The default options under "Update" and "Scan type" should stay selected.

The Scan type Option

The **Scan type** option allows you to specify how exhaustive a job the `Update Statistics` task does. In order to do the most thorough job, and produce the most complete set of statistics, SQL Server will need to example every single row in every table on which the task is being run. This is what happens when you select the default option of **Full scan**.

The alternative is to select the **Sample by** option, and specify the percentage of rows that should be sampled in order to update the statistics. This option is faster, and requires fewer server resources, but may produce statistics that are incomplete or inaccurate, causing some queries to perform less than optimally.

The hard part is in determining which option to choose. If there is a wide distribution of data in a given database, then the optimizer will need to sample all or most of the rows in order to generate accurate statistics. If the data in the rows varies little, then sampling only some of the rows will generally produce good enough statistics to create optimal query plans.

If you're using T-SQL or PowerShell scripts, you can work this out on a table-by-table basis and configure your scripts appropriately. However, since we're using the Maintenance Plan Wizard and are striving for simplicity, we have to compromise.

My recommendation is to use the **Full Scan** option whenever you run the `Update Statistics` task, except perhaps in those cases where your databases are very large. If your databases are large, and if using the **Full Scan** option causes performance problems because of the resources needed to perform the **Full Scan**, then use a **Sample by** scan to reduce the time and resources it takes to perform this task. Start out at 50%, and see how much you save in time and resources, and how reducing the sampling rate affects query performance. You may have to conduct several experiments to find the idea sample rate to best meet your needs.

Creating the Job Schedule

The last option to configure is the job schedule. As discussed previously, you should generally only schedule the `Update Statistics` task if you run the `Reorganize Index` task, and you should run one immediately after the other, but not overlap the two jobs. The exception to this is if you only rebuild or reorganize indexes once a week and you want update statistics on the nights when the indexes aren't being rebuilt or reorganized.

In our example, we want to run the `Update Statistics` task immediately after the `Reorganize Index` task, which takes place at 2 a.m. on Sunday, during our maintenance window (see Figure 7.10). So, depending on how long the `Reorganize Index` task takes, we may decide to schedule the `Update Statistics` task for, say, 4 a.m. as our best guess, and then adjust it accordingly, once we learn how long the `Reorganize Index` task really takes.

Summary

In theory, index and column statistics should update themselves but, as we have seen in this chapter, this is not always the case. For example, if we run the `Reorganize Index` task, we need to manually update statistics. Or, under some circumstances, where statistics become out of date sooner than expected, causing some queries to perform erratically, we may need to run the `Reorganize Index` task more often. If you don't have a good understanding of how statistics work, it is worth your while to learn more about them, as they play a large part in how well your SQL Server instance performs.

In the next chapter, we learn about the `Execute SQL Server Agent Job` task, which allows us to run a job from within a Maintenance Plan.

Chapter 10: Execute SQL Server Agent Job Task

The `Execute SQL Server Agent Job` task does exactly what it says: it allows you to run one (and only one) predefined SQL Server Agent job as part of a Maintenance Plan created with the Maintenance Plan Wizard.

Why would you want to run a SQL Server Agent job as part of a Maintenance Plan? In most cases, you probably wouldn't. As I have mentioned before, the biggest benefit of using the Maintenance Plan Wizard is simplicity, and if its simple maintenance model doesn't work for your environment, then you are probably often better off using T-SQL or PowerShell scripts to perform maintenance tasks, as they are more powerful and flexible than what a single SQL Server Agent job can do for you.

However, if you happen to want to run a single, simple SQL Server Agent job as part of your Maintenance Plan, it is something that you can do. For example, you might want to run a nightly job to check how much disk space is left on your drives and, if the amount is less than 20%, to have the job send you a warning message.

An Overview of the Execute SQL Server Agent Job Task

SQL Server Agent is a Windows service that you can use to execute scheduled maintenance tasks, which are called **jobs**. As we discussed in Chapter 3, when you create a Maintenance Plan using the wizard, SQL Server implements it as an SSIS package and, under the covers, creates the number of SQL Server Agent jobs required run the tasks in the plan.

In addition to the ten defined tasks that the Wizard allows you to configure and schedule, the `Execute SQL Server Agent Job` allows you to add one additional "custom" maintenance task to a given Maintenance Plan. This custom task is defined as part of the Agent job, not in the Wizard, so in effect all you are doing is adding a predefined job to the plan, and scheduling when it should run.

The nature of this task depends entirely on the nature of the predefined SQL Server Agent job, and these jobs can be used to perform many different tasks.

For example, you may have a custom job that:

- checks disk space and sends you a warning if it's getting near full
- kicks off a SQL Trace script to capture trace data on a scheduled basis
- checks a particular value, or values, in a DMV that you are interested in monitoring, and sends you an alert if the value(s) exceeds a predefined threshold
- starts a job that copies local `MDF` and `LDF` backups from off the local SQL Server instance to another server location (preferably offsite).

As discussed in the introduction, if you find yourself wanting to add multiple SQL Server Agent jobs to the plans you're creating using the Wizard, you're probably better off avoiding the Wizard in the first place, and using T-SQL or PowerShell scripting.

When the `Execute SQL Server Agent Job` task runs, T-SQL similar to the following is executed.

```
EXEC msdb.dbo.sp_start_job @job_id=N'cb73ea96-9a96-49fe-ada9-
a70a941f9fb9'
```

Notice that this is the execution of a system-stored procedure, which is instructed to run a job with a specified internal number. The number is not very useful to us, but if you want to look it up, in order to find out exactly what job was run (assuming you don't know), you could run the following `SELECT` statement:

```
SELECT * FROM msdb.dbo.sysjobs_view
```

This query will display all job IDs, along with their job names, so you can easily identify which job ID matches which job name.

When an `Execute SQL Server Agent Job` task runs, it produces a text report similar to the following:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: MaintenancePlan
Duration: 00:00:00
Status: Succeeded.
Details:
Execute SQL Server Agent Job (HAWAII)
Execute Job on Local server connection
Job name: Send Alert If Disk Space Exceeds 80% of Capacity
Task start: 2009-07-30T16:06:17.
Task end: 2009-07-30T16:06:17.
```

```
Success
Command:EXEC msdb.dbo.sp_start_job @job_id=N''cb73ea96-9a96-49fe-
ada9-a70a941f9fb9''
GO
```

Given that the `Execute SQL Server Agent Job` task can only run a single job, this report is not very long. You may want to note that the name of the job is included in the report, which can make it easier to troubleshoot potential problems with this task, should one arise.

When and How Often to Run the Custom Job

When and how often you run this job will depend on what the SQL Server Agent job does. If you are running a very lightweight job, such as checking disk space, you can run it almost whenever you want, and as often as you want. On the other hand, if the SQL Server Agent job uses a lot of server resources to run, and/or takes a long time to run, then you will have to schedule the job so that it doesn't interfere with users accessing the database, or with the running of other jobs. Ideally, you will schedule it during available maintenance windows.

Creating SQL Server Agent Jobs

If you decide to use the `Execute SQL Server Agent Job` task, you'll first need to create and configure the custom SQL Server Agent job that you want to run as part of your plan. How to create a SQL Server Agent job is beyond the scope of this book, but is explained in Books Online.

You'd create the job just like you would any other SQL Server Agent job, except that you won't schedule it, as you will use the Maintenance Plan Wizard to do the scheduling of the job for you. In addition, you will want to ensure that the SQL Server Agent job has been properly created and works as expected before you add it to a Maintenance Plan.

Configuring the Execute SQL Server Agent Job Task

The **Define Execute SQL Server Agent Job Task** screen is very straightforward. Your only options are to choose the single job that you want to execute as part of the plan, and schedule when the job is to run.

Selecting the Job

Figure 10.1 shows some the available SQL Server Agent jobs on my server. Every SQL Server Agent job on the server will be displayed, and you may have to scroll down to see them all. If you don't see any SQL Server Agent jobs listed in the Wizard, then you haven't created any, and should refer to the previous section!

Notice, in Figure 10.1, that there are checkboxes next to each job, giving the impression that you should be able to select multiple jobs from this screen and run them as part of your plan. This is not the case. If you select one checkbox, then another, the first checkbox you selected is deselected and only the new one remains selected. If Microsoft was following its own user interface design guidelines, there would be radio buttons here instead of checkboxes.

For this example, I've selected the **Send Alert If Disk Space Exceeds 80% of Capacity** job, as the SQL Server Agent job I want included as part of my Maintenance Plan.

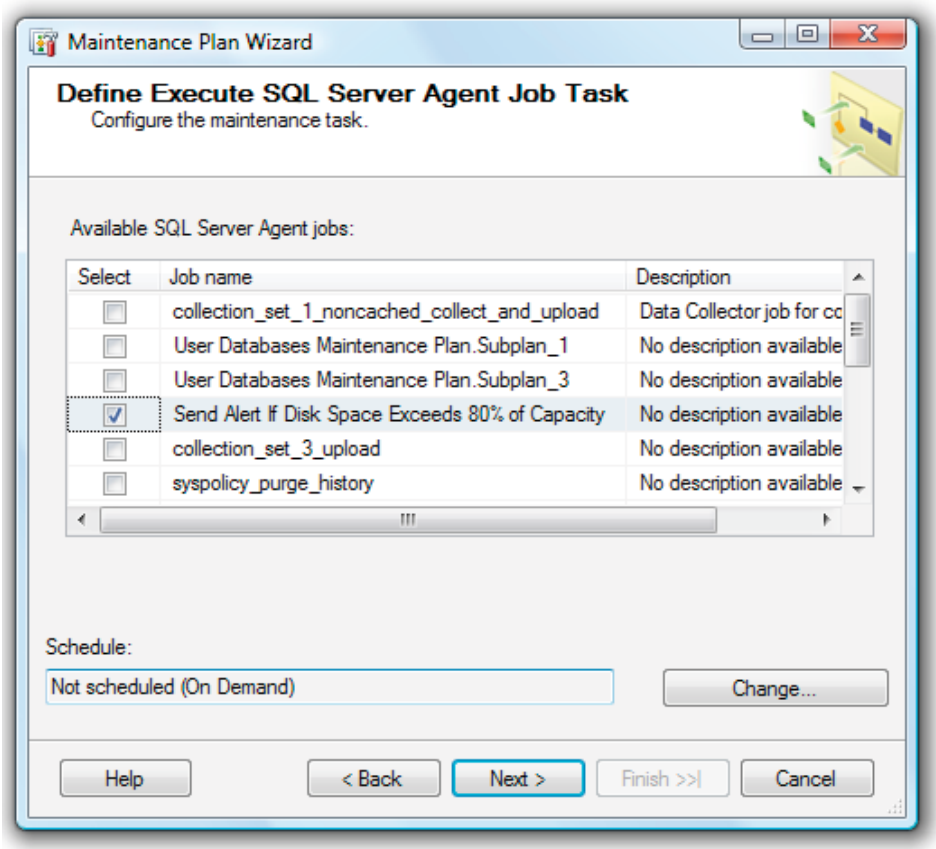

Figure 10.1: You can only select one SQL Server Agent job to execute as part of a Maintenance Plan.

Creating the Job Schedule

The next, and last step, as always, is to schedule when the task should run, using the **Schedule** option. Scheduling this task is like scheduling all the other tasks using the Maintenance Plan Wizard. Ideally, the job should run during a maintenance window, or at least during a slower time of the day, and it should not overlap other plan tasks. Your scheduling, of course, will have to take into account the resource impact of the SQL Server Agent job. If the job is lightweight, such as checking disk space, then you have great flexibility when running it. But if the job is heavyweight, and uses a lot of SQL Server resources, then you will have to be much more careful about scheduling when, and how often, it runs.

Summary

In theory, the `Execute SQL Server Agent Job` task is designed to add a little bit of flexibility to Maintenance Plans created using the Maintenance Plan Wizard. As long as you keep any `Execute SQL Server Agent Job` task simple and lightweight, you shouldn't run into any problems. On the other hand, it is important not to misuse this feature, and try to make it perform tasks it is not really designed to do. If you need database maintenance functionality that does not exist inside the Wizard, then take my advice (which by now may seem to be a little repetitive), and consider performing your database maintenance using T-SQL or PowerShell scripts instead.

In the next chapter, we learn about the `History Cleanup` task, which performs an important function many DBAs forget they need to carry out.

Chapter 11: History Cleanup Task

The `History Cleanup` task is very straightforward. It simply removes old data from the `msdb` database, and that's it. Over time, as backup and restore jobs run, as SQL Server Agents jobs run, and as Maintenance Plans run, historical data about each of these jobs is stored in tables of the `msdb` database.

In the short term, data stored in `msdb` can be useful. For example, if you are having problems with SQL Server Agent jobs, or Maintenance Plans, then this past `msdb` data can be used to help troubleshoot what went wrong. Also, SSMS uses the data stored in `msdb` about backup jobs to make it easier for you to use SSMS to restore databases or log files. Of course, you can also restore backups and logs using T-SQL commands that don't require this data.

Ultimately, however, this data has a limited life span, and once it gets old, there is no point in keeping it around.

An Overview of the History Cleanup Task

The `msdb` database is often referred to as the "SQL Agent database," since SQL Server Agent uses it to store all sorts of information about the jobs it runs. Unfortunately, SQL Server doesn't do any kind of its own internal clean up of this data so, over time, the `msdb` database can grow, and grow, and grow. It can even lead to some minor performance problems. As such, the DBA is responsible for cleaning up old records from `msdb` that are no longer of any value, and the `History Cleanup` task is designed to make that job easy.

When the `History Cleanup` task runs using its default settings, it executes the following T-SQL code:

```
declare @dt datetime select @dt = cast(N'2009-07-22T14:19:13' as
datetime)
EXEC msdb.dbo.sp_delete_backuphistory @dt
GO
EXEC msdb.dbo.sp_purge_jobhistory  @oldest_date='2009-07-
22T14:19:13'
GO
EXECUTE msdb..sp_maintplan_delete_log null,null,'2009-07-
22T14:19:13'
```

As you can see, this task runs three different system-stored procedures, one each to clean up historical data for backups, jobs, and maintenance plans.

Each of the three stored procedures has the same value for the `oldest_date` parameter, which, by default, is set to a date four weeks older than the date the task is run. In other words, it cleans out any `msdb` data that is more than four weeks old. When a `History Cleanup` task runs, it produces a text report similar to the following:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: User Databases Maintenance Plan
Duration: 00:00:02
Status: Succeeded.
Details:
Clean Up History (HAWAII)
Cleanup history on Local server connection
History type: Backup,Job,Maintenance Plan
Age: Older than 4 Weeks
Task start: 2009-08-19T14:26:20.
Task end: 2009-08-19T14:26:22.
Success
Command:declare @dt datetime select @dt = cast(N''2009-07-
22T14:26:20'' as datetime) exec msdb.dbo.sp_delete_backuphistory
@dt
GO
EXEC msdb.dbo.sp_purge_jobhistory  @oldest_date=''2009-07-
22T14:26:20''
GO
EXECUTE msdb..sp_maintplan_delete_log null,null,''2009-07-
22T14:26:20''
GO
```

Besides the code that runs, one of the key things to look for in the text report is the **Age**, which indicates how many weeks' worth of data is being kept in `msdb`. If you want to change the default four-week value, you can, as we will shortly discuss.

When and How Often to Clean Up MSDB

This task uses very few resources, so it can be run any time you want, even during busy times of the day. I generally run it weekly, along with my other weekly jobs, although running it more often, or less often, won't really make much difference. Schedule it to run at your convenience.

Configuring the History Cleanup Task

The first thing to notice about the **Define History Cleanup Task** screen, shown in Figure 11.1, is that you don't have to select a database this time, because only the `msdb` database is affected by this task.

Selecting the Historical Data to Delete

Under **Select the historical data to delete** you have three options, which are all selected by default:

- Backup and restore history
- SQL Server Agent job history
- Maintenance plan history.

I don't know of any good reason not to regularly delete each of these types of historical data, so I recommend keeping all three options selected.

Next, you just need to specify an age beyond which historical data will be removed by this task. The default is to remove data that is more than four weeks old, and this is suitable for most systems.

If you want to keep the data around longer, that's fine, but I wouldn't keep it more than three months, as the older data doesn't serve any practical purpose.

Creating the Job Schedule

The last step is to set the schedule. Setting this job schedule for this task is identical to all the rest of the tasks. Because this job is lightweight, and because there is no point in running it daily, I generally schedule this job to run once a week, along with my other weekly jobs. While the order of when this job runs is not really important, I have traditionally run it near the end of my weekly jobs, along with the `Maintenance Cleanup` task, which performs a similar function. The `Maintenance Cleanup` task will be covered in Chapter 15.

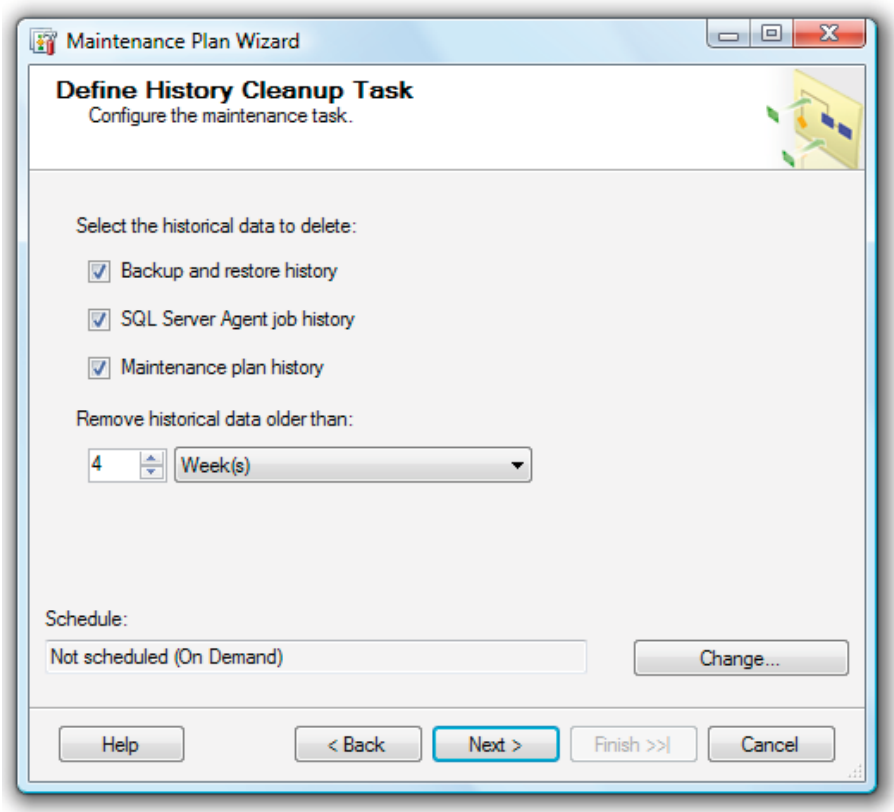

Figure 11.1: Completing the Define History Cleanup Task screen is fast and easy.

Summary

While the `History Cleanup` task might seem trivial in the larger scope of database maintenance, it still is important. For example, if you have a busy server, with lots of jobs running on it every day, the `msdb` database grows in size due to all the historical data it is storing, often leading to performance problems when SSMS accesses it. While this won't prevent you from doing your work, it can slow it down. And since most of the historical data is of no value, it might as well be cleaned out, helping `msdb` and SSMS to run more efficiently.

In the next three chapters, we learn about the three backup-related tasks available from the Maintenance Plan Wizard.

Chapter 12: Back Up Database (Full) Task

The basic purpose of a backup of a SQL Server database is to make a copy of the data so that it can be used to restore a database in the event of a disaster, such as system failure, damage to the database, corruption of the data it contains, and so on.

Of all the Maintenance Plan tasks that you need to perform, this is the most important. It is critical that you make full backups of your production databases on a regular basis, preferably once a day. A full backup can be performed on any database in SQL Server (expect for `tempdb`), no matter what recovery model it uses.

While the `Backup Database (Full)` task is the focus of this chapter, it is not the only backup task available from the Maintenance Plan Wizard. In the following two chapters, we will examine the `Backup Database (Differential)` task and the `Backup Database (Transaction Log)` task. In most cases, you will combine the `Backup Database (Full)` task with the `Backup Database (Transaction Log)` task in the same Maintenance Plan. In fact, once you have mastered the `Backup Database (Full)` task, you will find that the other backup tasks are very similar, and learning them will be easy.

Note, before we start, that the Maintenance Plan Wizard performs SQL Server **native** backups. If you want to perform a backup using a third-party tool then, in almost all cases, you will have to use the third-party's GUI tool, or T-SQL, to perform your backups. This is because third-party tools don't usually integrate with the Maintenance Plan Wizard. If you are using such a tool, you can still create a Maintenance Plan using the Maintenance Plan Wizard, but you will need to leave out the backup tasks that are included with it.

Backup Strategy – a Brief Primer

As I stated in the introduction, there are three basic types of database backups:

- Full backup – backs up all the data in the database. This is essentially making a copy of the `MDF` file(s) for a given database.
- Differential backups – a backup of any data that has changed since the last full backup. In other words, a differential backup makes a copy of any data in the `MDF` file(s) that has changed since the last full backup.

- Transaction log backups – a backup of the transaction log (`LDF`) file, which stores a history of the actions performed on the data since the last log backup (or database checkpoint if working in simple recovery mode). When a log backup is made, the live transaction log *generally* gets truncated to remove the backed up entries.

During both full and differential backups, enough of the transaction log is backed up to enable recovery of the backed up data, and reproduce any changes made while the backup was in progress. However, neither full nor differential backups truncate the transaction log.

All backups occur within the context of the **recovery model** of the database. The recovery model essentially dictates how the transaction log for that database is managed. The two main recovery models are described below (there is also a third model, **bulk logged** which, for the purpose of this discussion, we'll ignore but which is often used for bulk data operations).

- **Simple Recovery** – the transaction log is automatically truncated during periodic checkpoints and, because of this, it cannot be backed up and used to recover data.

- **Full Recovery** – the transaction log is not automatically truncated during periodic checkpoints and so can be backed up and used to recover data.

The backup strategy for a database in simple recovery mode relies entirely on full and differential backups. For example, you may take full backups of your simple recovery database every day or, for larger databases, you may opt to perform weekly full backups, and daily differential backups. An advantage of simple recovery mode is that you do not have to manage the transaction log and so the backup process is much simpler. On the down side, you are exposed to potential data loss to the extent of your last backup. In this example, that would be the potential loss of one day's data. The simple recovery model tends to be used for databases that are not "mission critical."

Transaction log truncation

There is often some confusion surrounding the topic of when the transaction log gets truncated, and a full discussion of the topic is outside the scope of this book. However, note the following:

1. *In simple recovery mode, the transaction log is automatically truncated whenever a checkpoint occurs.*

2. *In full (or bulk logged) recovery mode the transaction log is only truncated by a log backup. Truncation will take place as soon as the log backup is complete, assuming a checkpoint has occurred since the last log backup, and that there are not other factors preventing log truncation, such as a long-running transaction.*

3. *Truncating the log removes the backed up entries from the log, freeing up space to be reused. It does not physically shrink the log file.*

4. *Neither full nor differential backups truncate the transaction log.*

The backup strategy for a database in full recovery mode relies on both full (plus, if necessary, differential) backups and transaction log backups. So, for example, for a database in full recovery mode, you might perform daily full backups, and then log backups every hour, or perhaps even more frequently. During recovery, the full backup is restored, followed by the subsequent chain of transaction log backups. This process "rolls forward" all changes recorded in the transaction log backups and applies them to the full backup, thus making it possible to restore data up to the point in time of the disaster (or close). In this way, your exposure to potential data loss is minimized (in this case to an hour or less), but the administrative overhead to maintain this process is much larger.

Avoiding differential backups

For reasons that I'll explain as we progress, I tend to avoid differential backups if I can, as they make the backup process more complicated. My backup strategy, wherever possible, relies on full and transaction log backups only, as appropriate.

As noted at the start of this chapter, the reason to back up a database is in order to be able to restore it and recover the data it contains, in the event of a disaster. The type and frequency of backups that are suitable for each of your databases will be driven by business decisions that are made regarding the organization's tolerance to the potential loss of data from each database in question.

A full discussion of backup planning and strategy is outside the scope of this book, and will include consideration of type and frequency of backups as well as where they are stored, how they are tested, security, and so on. In fact, it's the topic for a short book in its own. A good place to start is the *Introduction to Backup and Restore Strategies in SQL Server* paper on MSDN, but there are many other resources available. However, hopefully this primer has provided enough background to work through this and the two subsequent chapters.

Suffice it to say, before we move on, that just because you have obtained a full database backup, does not necessarily mean that you can restore it. The only way to ensure that your backups are sound, and the data they contain recoverable, is to test them!

An Overview of the Backup Database (Full) task

The purpose of the `Backup Database (Full)` task is to perform a full backup of the selected database(s). When the `Backup Database (Full)` task runs using its default settings, it executes the following T-SQL code (in this example, we're performing a full backup of the `AdventureWorks` database):

```
BACKUP DATABASE [AdventureWorks] TO DISK = N'C:\Program
Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Backup\
AdventureWorks_backup_2009_08_19_145336_3160000.bak'
WITH NOFORMAT, NOINIT, NAME =  N'AdventureWorks_back
up_2009_08_19_145336_3150000', SKIP, REWIND, NOUNLOAD,  STATS =
10
```

This is the standard `BACKUP DATABASE` command where the backed up database is being written to disk in a specified folder. Ideally, this disk will be on a locally attached drive, a SAN, or a NAS device that has been designated specifically for storing backups.

The backup file for a Full (and Differential) backup uses the `BAK` extension. The name of the backup is the name of the database appended with a time stamp that uniquely identifies that backup. This makes it easy for you to identify backups when you view them inside a folder. When a `Backup Database (Full)` task runs, it produces a text report similar to this one:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: User Databases Maintenance Plan
Duration: 00:00:04
Status: Succeeded.
Details:
Back Up Database (Full) (HAWAII)
Backup Database on Local server connection
Databases: AdventureWorks
Type: Full
Append existing
Task start: 2009-08-19T15:02:10.
Task end: 2009-08-19T15:02:14.
Success
Command: BACKUP DATABASE [AdventureWorks] TO  DISK = N''C:\
Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\
Backup\AdventureWorks_backup_2009_08_19_150210_7570000.
bak'' WITH NOFORMAT, NOINIT,  NAME = N''AdventureWorks_back
```

```
up_2009_08_19_150210_7560000'', SKIP, REWIND, NOUNLOAD,  STATS =
10
GO
```

Since, in this example, I was only backing up the `AdventureWorks` database, this is the only database I see in this report. If I backed up other databases, then you would see a separate `BACKUP DATABASE` command for every database backed up by this task.

Append existing

You might notice the phrase "Append existing" in the previous report. Ignore it, as it is only relevant if you back up to a backup device or stripped set, which is something I will discuss a little later, and which I recommend you generally avoid.

When and How Often to Perform Full Backups

Here's my backup philosophy. My goal is to try and perform a full backup of all my databases **every night**. In most cases, taking a full backup incurs minimal overhead on your SQL Server, although I still prefer to run full backups when servers are less busy (usually at night). One exception to this would be if your SQL Server already has an I/O bottleneck; in this case, performing a backup at most any time will increase disk contention and hurt server performance. Of course, if you have this problem, then you should consider tweaking the performance of your queries, or getting a faster disk I/O subsystem.

If your databases are very large, you may find that you do not have the available resources (such as storage space), or even enough time, to back them up every night. One option that is worth considering, if you find yourself in this situation, is purchasing a third-party backup tool. Such tools provide high-speed, compressed, and encrypted backups that you can fit into less disk space and smaller maintenance windows. As noted in the introduction to this chapter, this will almost certainly mean that you cannot use the Maintenance Plan Wizard to schedule your backups, and instead you should use scripting, or the facilities provided by the third-party tool, to perform the backups.

Backup compression in SQL Server 2008 Enterprise Edition

SQL Server 2008 Enterprise Edition (not standard edition) offers built-in backup compression, but it does not offer backup encryption. We will discuss this feature later in this chapter.

For the typical database, using one of the above strategies will allow you to perform full backups nightly. If your database is so large that neither of these two options will work, then you probably shouldn't be using the Maintenance Plan Wizard to perform your backups. Backing up large databases (100 GB or more), often requires more sophisticated backup strategies, which are best handled using third-party tools, T-SQL or PowerShell scripts.

If your databases are large, and if scripting and third-party tools are not an option, then you may need to consider running full backups less frequently, and performing differential backups (covered in the next chapter) on the days in between.

Configuring the Back Up Database (Full) Task

The **Define Back Up Database (Full) Task** screen, shown in Figure 12.1, is more complex than most of the Maintenance Plan Wizard screens we have seen up to this point.

The first thing I want to point out is that the **Backup type** drop-down, at the very top of the screen, contains the value **Full**, and is grayed out. That's because we are on the `Back Up Database (Full)` task, and the Wizard is smart enough to complete the option for us.

Database and Backup Component Selection

The next option is the **Database(s)** drop-down box that we've seen many times before, and which you use to select the databases that you want to subject to full backups. In previous chapters, we've often seen **Object** and **Selection** options presented here, allowing us to narrow the scope of a given task to defined objects within the database.

In essence, the **Backup component** options, below the **Database(s)** box (both of which are currently grayed out) offer a comparable service for backups. When these options are not grayed out (more on this shortly) you can either select **Database** to back up the whole database, which you will almost always want to do, or you can select **Files and filegroups** to back up only specific files or filegroups within that database.

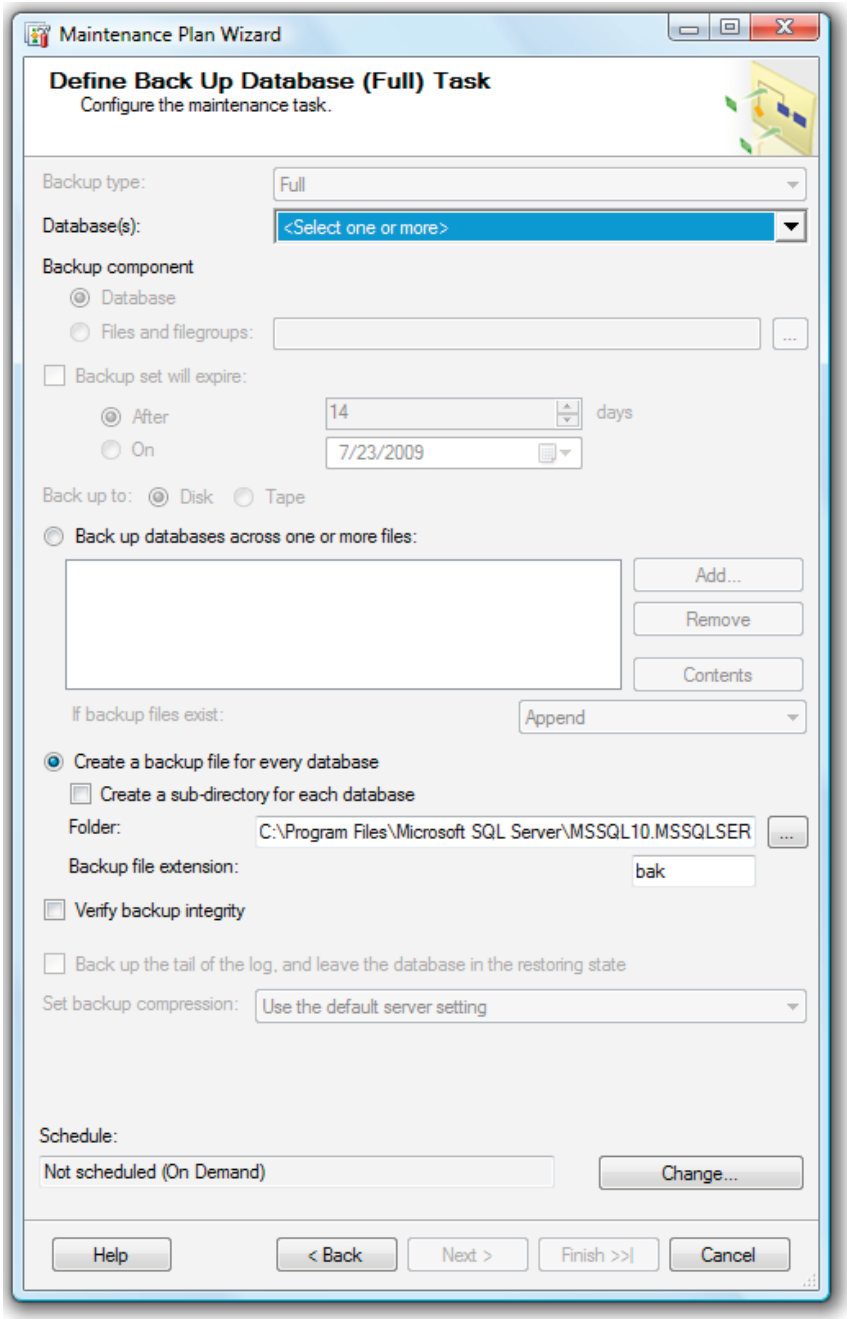

Figure 12.1: **This is the most complex screen of the Maintenance Plan Wizard we have seen so far.**

Let's take a look at this in a little more detail. In order to be consistent with the previous tasks, we would normally want select **All user databases**. However, in order to demonstrate the next feature of this task, let's instead select a single database, `AdventureWorks`, as shown in Figure 12.2, and click on **OK**.

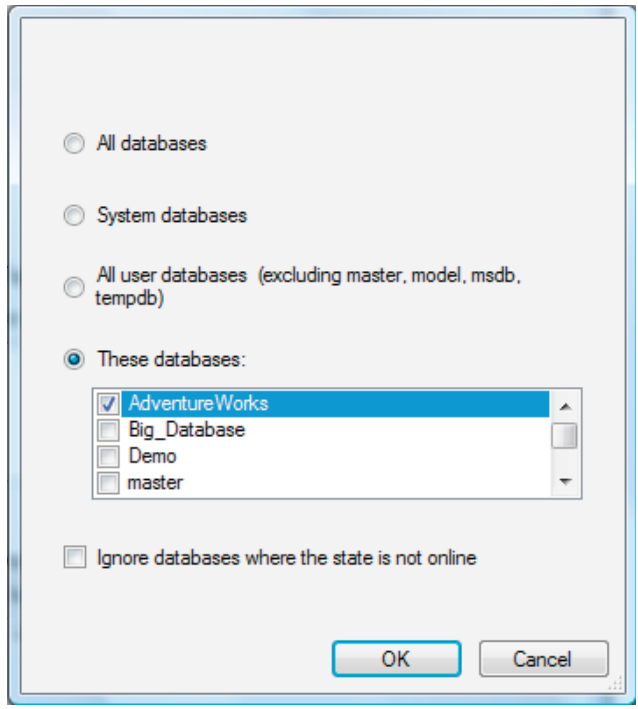

Figure 12.2: The two options under "Backup component" will only be available if you select a single database.

Now, under **Backup component**, you'll see that **Database** is available and selected, but **Files and filegroups** is grayed out, as shown in Figure 12.3. This means that the whole database will be backed up. If we had not selected a single database, then both **Database** and **Files and filegroups** would still be grayed out. This is because these two options only work if a single database has been selected.

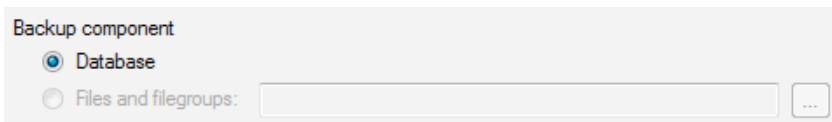

Figure 12.3: You have two choices, either to perform a full backup or, if a database has multiple files or file groups, to back up only a portion of the database.

The **Files and filegroups** option only becomes available if a single database is selected and is composed of multiple files or filegroups, which **AdventureWorks** is not. However, if it were, you'd be able to select that option, and so choose to only perform a full backup of specific files or filegroups, selected using the browse button. This option is sometimes useful for large databases but I would say that, if your databases have multiple files or filegroups, then you should probably not be being using the Maintenance Plan Wizard for this task. Sure, it will do it for you, but if you are at that level of database complexity, you should really be using T-SQL or PowerShell scripts to perform this task.

Now, let's get back to the practical use of the Maintenance Plan Wizard. Go back and choose our usual option of **All user databases**, as shown in Figure 12.4. At this point, both of the **Backup component** options are grayed out, so we'll automatically be performing full backups of the whole of each of the user databases.

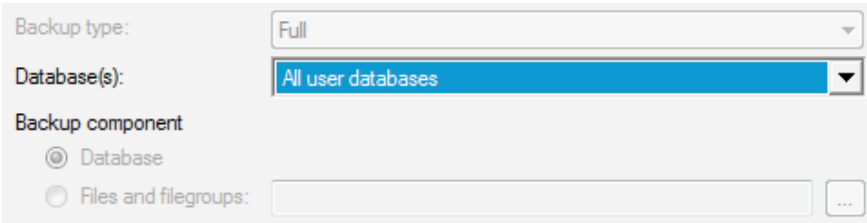

Figure 12.4: When you choose two or more databases to back up, the two "Backup component" options will not be available.

The next available option on the screen is **Backup set will expire**. It is not selected by default, and the two options below it are grayed out, as shown in Figure 12.5.

Figure 12.5: Assuming you create backups sets, you can choose when they expire.

This option determines when a backup set can be overwritten, and it is designed for people who back up their databases directly to tape. Virtually nobody backs up a SQL Server database directly to tape any more, so you will not need to select this option.

The next option is **Back up to**, where you can choose between backing up to disk, or to a directly attached tape drive, as shown in Figure 12.6.

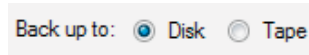

Figure 12.6: You will always choose Disk.

As discussed, the only real option is to back up to disk, so leave this option set to **Disk**.

Don't back up directly to tape

SQL Server supports the option to directly back up a database to a direct attached tape drive (the tape drive is physically attached to the SQL Server), but this option is virtually never used. Why? Backing up to a tape drive directly from a single server is expensive (you need a tape drive for every server), difficult to administer, unreliable, and slow. Today, virtually all databases are backed up to disk first, then either stored on a SAN, NAS, in centralized tape library or, in some cases, Internet backup services.

Backup File Storage

In some ways, the next section of the screen, shown in Figure 12.7, forms the heart of the backup task, since it allows us to define how and where the backup files are stored.

Let's look at each option in turn.

Backup databases across one or more files

If you choose this option, you can either **back up to a backup device**, which is a pre-created file that can hold one or more backups, or **create striped backups**, which allow you to perform a backup of a database on two or more physical files at the same time.

Backup devices are a holdover from previous versions of SQL Server and are no longer used much by DBAs. They are hard to work with and don't offer many advantages over standard backup files, which we will discuss in the next section.

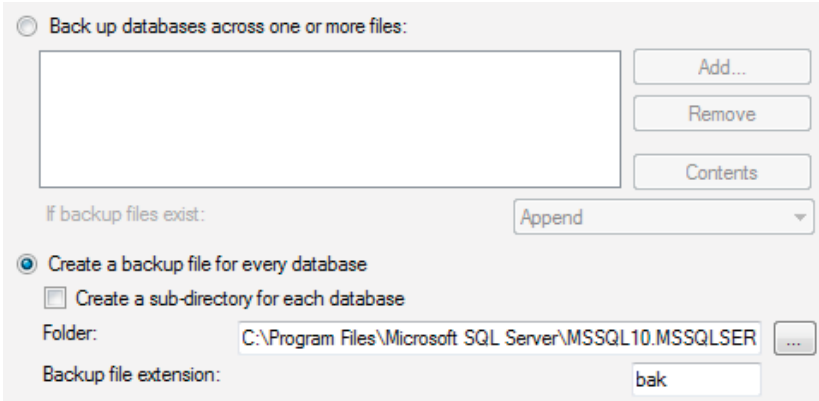

Figure 12.7: Generally, the second option, selected by default, is the choice to make.

A striped backup can, under certain conditions, speed up the backup process. However, like backup devices, this option is not used very often, because there are many better ways to boost backup performance that are faster, save disk space, and are easier to administer. This better way, backup compression, we will discuss a little more in the pages ahead.

As you have probably already guessed, I recommend that you don't use this option in your Maintenance Plan. If, for some reason, you do need it, you may be better off using T-SQL or PowerShell scripts instead, as trying to use this option via the Maintenance Plan Wizard is tedious and not very flexible.

Create a backup file for every database

This is the default option, and the one you should select. It will automatically create a new backup file on disk for each database you selected in the **Database(s)** section of the screen. Backup files will automatically be assigned the name of the database, along with the word **backup:** and the date of the backup. This means that you can easily identify which backup file is which, and when it was taken. This option has three of its own sub-options. Let's look at them one at a time.

Create a sub-directory for each database

If you select this option, a sub-directory will be created for each database you back up, and the associated backups will be stored there. So, if you back up two databases, one called `Sales` and one called `Marketing`, then backups for the `Sales` database will be stored in a sub-directory called **Sales** and those for `Marketing` database in a sub-directory called **Marketing**.

If you don't choose this option, then all the backups will be stored in the same folder (which you specify with the next option). Either option works fine. Personally, I don't use sub-directories because I don't like to go through multiple levels of folders to view my backup files. However, other DBAs like the organization provided by using this option. The choice is yours.

Folder

This defines the parent folder that will used to store all database backup files arising from execution of this task. The choice of folder is a very important decision, and you should not automatically select the default folder to store your backups.

Ideally, you will have a destination designated specifically for storing backups, on a locally attached drive, SAN, or NAS device. Your backups should not be stored on the same drive locations as your "live" `MDF` and `LDF` files, otherwise you might experience I/O contention, when backups are made, that could affect the performance of your servers. Use the browse button to select the backup location.

> ***Backing up over a network***
>
> *You can also perform backups over the network using this option. This is not my personal preference because, if you have a network problem while the backup is being made, the backup will fail, and the Maintenance Wizard doesn't provide a way to recover. If you have to back up over the network, then you need to consider a third-party backup tool that is designed for network resilience; or to create your own T-SQL or PowerShell script that can detect a failed backup and then restart it once the network is back up and running.*

Backup file extension

The default backup file extension is `BAK` and it should not be changed. If you change it, you risk confusing yourself, and others, about which files are backups and which are not.

Verify backup integrity

The next option is **Verify backup integrity**, as shown in Figure 12.8.

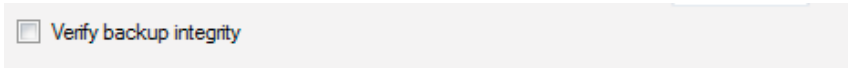

Figure 12.8: I strongly suggest that you always select this option.

When this option is selected, the `RESTORE VERIFYONLY` command will be run against the completed backup. This command performs multiple checks on the backup to test that the backup is complete and readable. While the command does not verify the structure of the data in the backup (that is what the `Check Database Integrity` task is for, see Chapter 5), it does a very good verification of the backup and, if it passes the verification, you can be fairly certain the backup is a good one.

Don't rely on the Verify backup integrity option alone!

It is a very good test, but it is still not perfect. The only way to verify that you have a good backup is to perform a restore and see if you can read the data. Ideally, you should randomly select backups on a regular basis, and perform a test restore on them to ensure that your backup and restore process is working as expected.

Next on the screen is the option **Back up the tail of the log, and leave the database in the restore state**, which is grayed out. This is normal, as this is a generic screen (and code) that is used in other parts of Management Studio, and it is not applicable to the creation of a Maintenance Plan.

Set backup compression

The Set backup compression option is only available if you have the Enterprise Edition of SQL Server 2008. It is not available with the Standard Edition of SQL Server 2008 or with any edition of SQL Server before 2008.

Third-party backup compression

If you are running Standard Edition, or an older version of SQL Server, but would like the ability to compress your backups, you can. You just need a third-party application, such as Red Gate's SQL Backup. Purchasing a third-party backup application is a lot less expensive than purchasing the Enterprise Edition of SQL Server. In addition, the backup compression included with the Enterprise Edition of SQL Server 2008 does not perform backup encryption, which means that your backups are not protected from prying eyes.

If you do have SQL Server 2008 Enterprise Edition, you have the ability to make one of three choices, as shown in Figure 12.9.

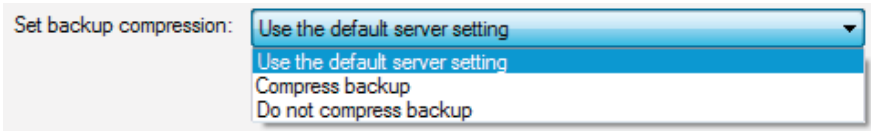

Figure 12.9: The backup compression option offers three choices.

The first option, **Use the default server setting**, specifies that the backup file should be compressed using whatever is the default backup compression setting for your server. This is a `sp_configure` option where the `compression default option` server option is set to either 0, which is off, or 1, which is on. By default, the server-side default backup compression setting is turned to 0.

The second option, **Compress backup**, turns backup compression on and overrides the default setting for the `compression default option` server option. This is the most likely option to choose, as it will guarantee that all the databases backed up by this Maintenance Plan will be compressed.

The third option, **Do not compress backup**, tells the Maintenance Plan Wizard not to use backup compression. If you have the `compression default option` server option set to 1, which means that backup compression is turned on, then selecting this option will override the server-wide setting, and the database backup will not be compressed. This option is not really applicable to Maintenance Plans, and should not be used.

Creating the Job Schedule

Finally, we arrive at the **Schedule** option, with which we are already familiar. In our running example in this book, most database maintenance is performed during the Sunday maintenance period. For this task, though, we want to perform a full backup every day. To create a daily schedule, there are two parts of **the Job Schedule Properties** screen to which we need to pay particular attention.

First, under **Frequency**, set the **Occurs** drop-down box to **Daily**, ensuring that a full backup will be made daily. Second, under **Daily frequency**, select **Occurs once at:** and then enter the time of the day when the full backup is to be made. In this example, the full backup will be made at 5 a.m. All the other options on the screen can remain at their default values, and the final screen should look as shown in Figure 12.10.

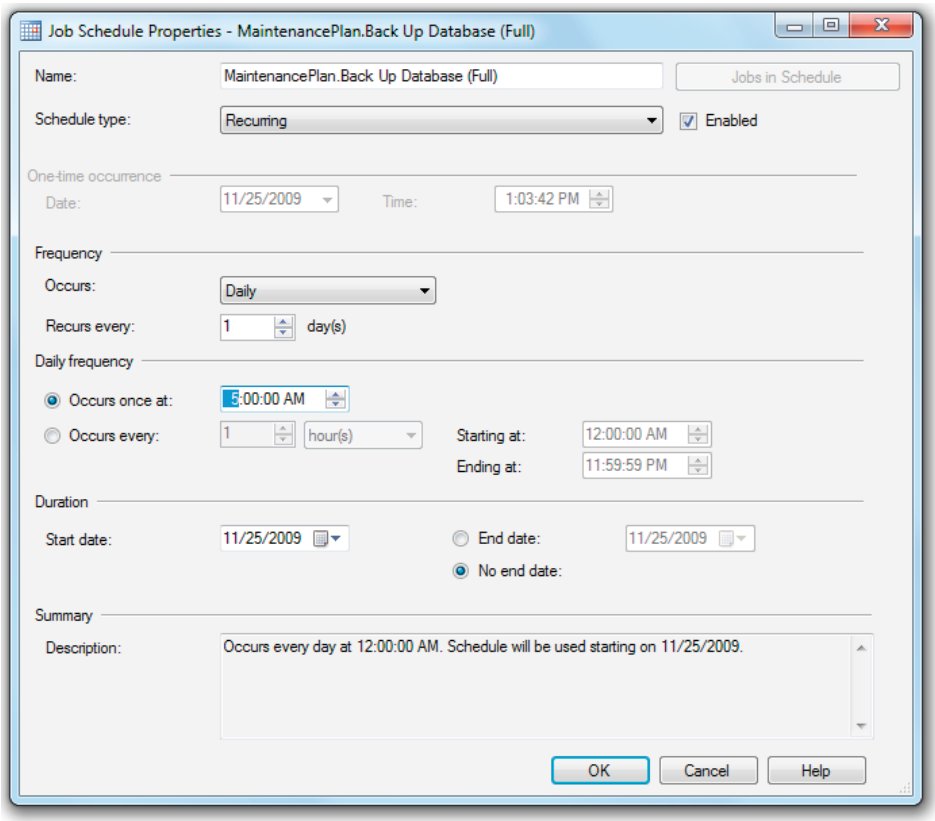

Figure 12.10: The above schedule performs a full backup every day.

Why select 5 a.m. to perform the full backups? This brings us to a particularly dicey issue. Before I begin, I first want to repeat something I have said many times already in this book: the purpose of using the Maintenance Plan Wizard is to keep database maintenance as *simple* as possible. With this philosophy in mind, I choose 5 a.m. because it falls after the completion of the last weekly job that is performed on Sunday.

Unfortunately, neither the Maintenance Plan Wizard nor SSMS gives us an easy way to see our entire job schedule in a single view. This means that you have to take extra care, when scheduling daily jobs, to make sure they don't interfere with weekly jobs.

One of the last tasks we scheduled to run in the weekly maintenance window, on Sunday, was the `Rebuild Index` task, or the `Reorganize Index` task plus the `Update Statistics` task. Whichever option is chosen to remove index fragmentation, we need to wait until that task is complete before we schedule the daily `Backup Database (Full)` task, in order to prevent the jobs from overlapping. By scheduling a full database backup time of 5 a.m. we are making the assumption that our index defragmentation job will have been completed by then, and

also that the full database backup job will be completed before people come to work in the morning. As you can see, there are a lot of assumptions being made, and this is why I referred above to this discussion as being dicey.

As discussed in Chapter 4, when you schedule jobs using the Maintenance Plan Wizard, you have to take an initial guess as to when a job should run so that it doesn't interfere with other jobs. Once you have tested your jobs, and also run them for a few days, you will get a better feel for how good your assumptions were. If they were wrong, you will then have to change the various scheduled jobs so that they "fit" better together. You will learn how to change a Maintenance Plan schedule in Chapter 19, on the Maintenance Plan Designer.

Another thought may have occurred to you during this discussion: is it possible to schedule the daily full backup job at a specified time Monday through Saturday, and then at a different time on Sunday, so as not to interfere with the weekly Sunday maintenance jobs? Yes, it is, but it is at this point that the Maintenance Plan Wizard falls short, as it does not allow you to set multiple schedules for a single task within the same Maintenance Plan. The work-around for this is to create multiple Maintenance Plans, each with their own independent schedules. Another option is to use the Maintenance Plan Designer, which allows greater flexibility than does the Wizard.

In other words, what I am saying is that, if you want to perform tasks that are not easily done from within the Wizard, such as creating multiple schedules for the same task, then you should probably consider using the Maintenance Plan Designer to create your Maintenance Plans, or using T-SQL or PowerShell scripts instead.

Summary

Of all the maintenance tasks you perform, the most critical one is the `Backup Database (Full)` task. Performing regular full backups is the only way you can guarantee that your data is protected should the original database become unavailable. On the other hand, the `Backup Database (Full)` task is not the only backup task you need to perform regularly. In the next two chapters, we will take a look at two other backup-related tasks.

Chapter 13: Back Up Database (Differential) Task

As you can guess by its name, this task is designed to create differential backups, in other words, to back up all changes in a database since the last full backup.

Performing differential backups is not a substitute for making regular transaction log backups. If your databases are in the full recovery mode, transaction log backups must be included in your overall backup strategy alongside full backups and any differential backups.

For reasons that will become clear as we progress through the chapter, I generally avoid performing differential backups if I can, preferring to rely on a combination of full and transaction log backups. Furthermore, if your database maintenance requirements do include performing differential backups, then I recommend that you create your Maintenance Plans using the Maintenance Plan Designer (see Chapter 16 onwards), rather than the Wizard.

An Overview of the Back Up Database (Differential) Task

Since differential backups only copy the data that has changed since the last full backup (called the *differential base*), a differential backup is smaller than a full backup, so the nightly backup process is faster. However, as each day of the week passes, the differential backup will take longer to perform, as more and more data changes in the database.

When the `Backup Database (Differential)` task runs using its default settings, it executes the following T-SQL code (assuming `AdventureWorks` is being backed up):

```
BACKUP DATABASE [AdventureWorks] TO DISK = N'C:\Program
Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Backup\
AdventureWorks_backup_2009_08_19_154600_2850000.bak' WITH
DIFFERENTIAL, NOFORMAT, NOINIT, NAME = N'AdventureWorks_back
up_2009_08_19_154600_2830000', SKIP, REWIND, NOUNLOAD, STATS = 10
```

This `BACKUP DATABASE` command is virtually identical to the one we saw in the last chapter, for the `Backup Database (Full)` task, the only real difference being the addition to the command of the `DIFFERENTIAL` keyword.

Unfortunately, the name assigned to the differential backup file looks exactly like the name of

a full backup (other than the date stamp), making it difficult to distinguish full backups from differential backups. To keep track, take note of the time stamp for the differential base file (last full backup) and then identify subsequent differential backup files from there.

When a `Backup Database (Differential)` task runs, it produces a text report similar to the following:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: MaintenancePlan
Duration: 00:00:00
Status: Succeeded.
Details:
Back Up Database (Differential) (HAWAII)
Backup Database on Local server connection
Databases: AdventureWorks
Type: Differential
Append existing
Task start: 2009-08-19T15:54:39.
Task end: 2009-08-19T15:54:39.
Success
Command: BACKUP DATABASE [AdventureWorks] TO  DISK = N''C:\
Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\
Backup\AdventureWorks_backup_2009_08_19_155439_1450000.bak'' WITH
DIFFERENTIAL , NOFORMAT, NOINIT,  NAME = N''AdventureWorks_back
up_2009_08_19_155439_1450000'', SKIP, REWIND, NOUNLOAD, STATS =
10
GO
```

As you can see, the text report above is also virtually the same as the report for the `Backup Database (Full)`, other than the references to "differential." If more than one database were having a differential backup, then you would see each of them in the report.

When and How Often to Perform Differential Backups

Personally, I avoid differential backups, preferring to perform daily full backups, plus periodic transaction log backups. I find this process simpler and less confusing than combining full, differential, and transaction log backups as part of my backup plan.

Of course, there *are* reasons why you might want to incorporate differential backups in your Maintenance Plan. The most common reason given to use differential backups is when databases are very large, and taking a full backup each night is too time-consuming. What some DBAs do in these cases is to take a full backup once a week, and take differential backups for the remaining days of the week, while also incorporating transaction log backups. This reduces the amount of time each day it takes to perform a backup. On the other hand, as the week progresses, the differential backup time will get longer and longer, as data is changed in the database, negating some of the time savings provided by using differential backups. Only by experimenting will you know for sure if using differential backups is a good solution for your particular environment.

Configuring the Back Up Database (Differential) Task

Let's examine the **Define Back Up Database (Differential) Task** screen from the Maintenance Plan Wizard, shown in Figure 13.1.

The available options are the same as those seen and described for the `Backup Database (Full)` task in the previous chapter, so I'll refer you there for the details and will keep the explanation here short.

The **Backup type** option at the top of the screen is automatically filled in by the wizard, and displays **Differential**.

Database Selection and Backup Component

If you decide to perform differential backups, you'll have to give the following question some thought: do you want to perform differential backups on all the databases on your server, or just some of them?

In the context of the Wizard, differential backups can quickly trample over our goal of simplicity. Our aim throughout the book has been to apply each task to the same set of databases. So ideally, in the content of our running example, we'd select to apply differential backups to all user databases.

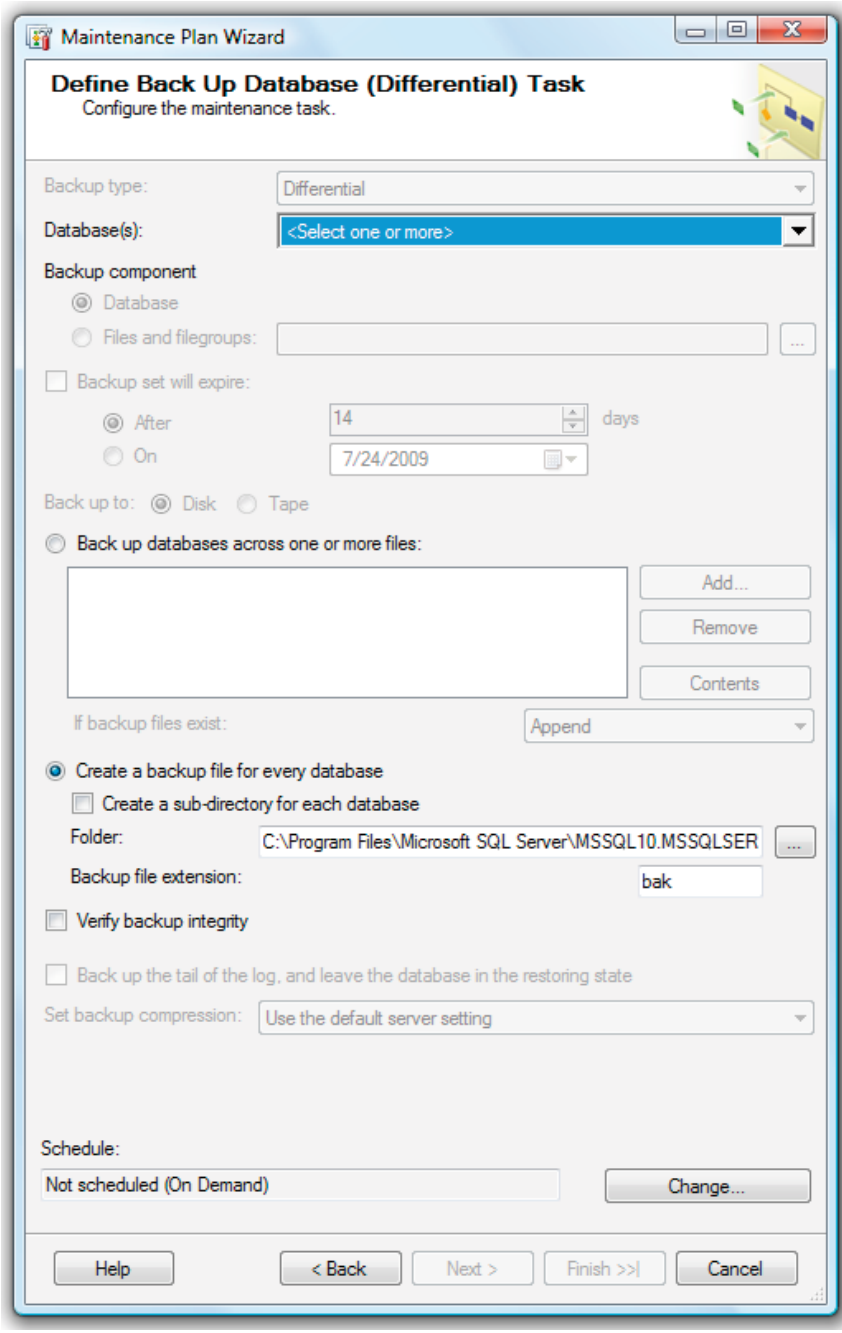

Figure 13.1: Creating and scheduling a differential backup task is virtually identical to taking a full backup.

On the other hand, if we want to perform differential backups on only some of our databases, then we will have to create multiple Maintenance Plans, adding some complexity to our database maintenance. This is because, as we have already learned, the Wizard does not allow us to include multiple instances of the same task within the same Maintenance Plan. At a minimum, we will have to create one Maintenance Plan for those databases that don't require differential backups, and one Maintenance Plan for those databases that do. Alternatively, we can use the Maintenance Plan Designer, which does allow multiple instances of the same task, as described in Chapter 16 onwards.

Other parts of the **Define Back Up Database (Differential) Task** screen are configured in exactly the same way as the **Define Back Up Database (Full) Task** screen, so I won't discuss them again here.

Creating the Job Schedule

If you choose to perform differential backups from within a Maintenance Plan created by the Wizard, you will have to carefully think through your backup schedule and how you want to do it. For example, you will have to create a task to perform a full backup once a week (see the previous chapter on how to do this), a separate task to perform daily differential backups, and another task to perform transaction log backups (more on this in the next chapter).

What can become a little confusing is how to schedule these three tasks so they don't interfere with each other. For example, let's assume that we want to perform a full backup on Sunday, differential backups on Monday through Saturday, and that we want to take transaction logs once each hour.

To do this, we would need to schedule these three tasks so that the full backup always occurs first on Sunday night; once a full backup is made, transaction log backups are made once an hour until Monday night, when the first differential backup is made. Once the Monday night differential backup is made, then we need to have transaction log backups made every hour until the Tuesday night differential backup, and so on, until we get back to the Sunday night full backup.

Scheduling all of these tasks within the Maintenance Plan Wizard is difficult, and if your schedules overlap, or are in the wrong order, your Maintenance Plan will most likely fail. If you need to perform differential backups then I think you will find the scheduling process less of a headache if you use the Maintenance Plan Designer instead, which offers much more flexibility.

Summary

The Maintenance Plan Wizard allows us to create differential backups, but using the `Backup Database (Differential)` task, via the Wizard, is not an easy proposition. My first choice is to avoid differential backups, and to keep my Maintenance Plans as simple as possible. On the other hand, if you need to create differential backups, then I recommend the use of the Maintenance Plan Designer, or the use of T-SQL or PowerShell scripts.

Chapter 14: Back Up Database (Transaction Log) Task

Earlier, I stated that the `Back Up Database (Full)` task is the most important maintenance task for DBAs to implement. Now we have come to the second most important task, the backing up of transaction logs, which is implemented using the `Back Up Database Transaction Log)` task.

In order to recover as much data as possible after the loss of a database, and to enable point-in-time recovery of your data, when using the full (or bulk-logged) recovery model, it is essential that you perform regular transaction log backups in addition to full backups (Chapter 12), and possibly differential backups (Chapter 13). The `Back Up Database (Transaction Log)` task allows you to schedule the backup of your transaction logs as part of the Maintenance Plan Wizard. This task is not relevant (or available) to any databases using the simple recovery model, as you cannot back up the transaction log of a database using this model.

Backing up a database's transaction log offers very important benefits. First of all, it makes a backup copy of all the transactions that have been recorded in the transaction log file since the last log backup. These backups, along with any available tail-log backups, help ensure that, should you have to restore your database, you are able to recover all of the data up until the point in time that the failure occurred (or very close to it). Essentially, the log backup files can be applied to a restored copy of a full database backup, and any transactions that occurred after the full backup will be "rolled forward" to restore the data to a given point in time.

The second important benefit of backing up your transaction log is that it truncates older data from a database's transaction log, which keeps your transaction log to a reasonable size. In fact, if you don't backup your transaction log, it can grow until you run out of disk space. Note, again, that only a transaction log backup, not a full or differential backup, will truncate a transaction log.

An Overview of the Backup Database (Transaction Log) Task

In order to perform transaction log backups, your databases must use either the full or the bulk-logged recovery models. If a database is set to the simple recovery model, then you cannot back up the transaction log, and it will automatically get truncated at regular intervals, when a database checkpoint occurs. Bear in mind that the `master` and `msdb` databases are set to the simple recovery model (which you should not change) and so you cannot back up their transaction logs.

A database's transaction log (`LDF`) file contains a list of entries describing all the actions performed on the data in that database. A transaction log backup makes a copy of the recorded actions in this file (the backup copy), which should be stored separately from the drive on which the live log and data files are stored. Backing up the transaction log generally results in truncation of the live log file, at which point the previously backed-up entries would be removed.

Transaction log backups are used in conjunction with full and differential backups to allow point-in-time recovery of data, in the event of disaster. In fact, it is not possible to perform a transaction log backup without first performing at least one full backup.

In the event of a disaster, for example a database becoming corrupt, the most current full backup is restored followed by the subsequent chain of transaction log backups. This process "rolls forward" the actions recorded in the transaction log backup files. In other words, it applies them to the full backup in order to rebuild the data as it existed at a certain point in time.

If the current transaction log is still accessible, which it may not be if, for example, the disk holding the transaction log has crashed, then the DBA can perform what is called a **tail-log backup**, in order to capture any actions recorded in the log that had not previously been backed up. The tail-log backup can be restored along with all of the other transaction log backups, minimizing the loss of any data.

Managing backups for databases in the bulk-logged recovery model

In almost all cases, you should use the full recovery model for your databases. The bulk-logged recovery model is typically only used during times when you are performing bulk operations. If you are managing backups for bulk-logged databases, then I advise you to use T-SQL or PowerShell scripts, rather than the Maintenance Plan Wizard, to perform your database maintenance.

When the Backup Database (Transaction Log) task runs using its default settings, it executes the following T-SQL code (assuming AdventureWorks is being backed up).

```
BACKUP LOG [AdventureWorks] TO DISK = N'C:\Program
Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\
Backup\AdventureWorks_backup_2009_08_20_111623_3462370.
trn' WITH NOFORMAT, NOINIT, NAME = N'AdventureWorks_back
up_2009_08_20_111623_3462370', SKIP, REWIND, NOUNLOAD, STATS = 10
```

The BACKUP LOG T-SQL command is similar to the BACKUP DATABASE command, but produces a transaction log backup file with the extension .TRN, as opposed to .BAK. The name of the database is used as part of the backup file name, along with an appropriate date stamp.

When a Backup Database (Transaction Log) task runs, it produces a text report similar to this one.

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: User Databases Maintenance Plan
Duration: 00:00:16
Status: Succeeded.
Details:
Back Up Database (Transaction Log) (HAWAII)
Backup Database on Local server connection
Databases: AdventureWorks
Type: Transaction Log
Append existing
Task start: 2009-08-20T11:11:39.
Task end: 2009-08-20T11:11:55.
Success
Command: BACKUP LOG [AdventureWorks] TO DISK = N''C:\
Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\
Backup\AdventureWorks_backup_2009_08_20_111139_2902370.
trn'' WITH NOFORMAT, NOINIT,  NAME = N''AdventureWorks_back
up_2009_08_20_111139_2892370'', SKIP, REWIND, NOUNLOAD,  STATS =
10
GO
```

While the above example only shows a single transaction log backup, all the transaction log backups you select will be shown in your text report.

When and How Often to Back Up Transaction Logs

Generally speaking, you want to back up your transaction logs often enough so that you are at minimal risk of losing data (you can't always count on being able to do a tail-log backup), and that the transaction log is kept down to a reasonable size (because backing up the transaction log truncates the transaction log).

The more frequently you take log backups, the smaller is your organization's exposure to the risk of potential data loss. On the downside, the more transaction log backups you take, the more administrative effort it requires, and the more server resources are used.

While there is no perfect transaction log backup interval, taking transaction log backups on an hourly basis is a fairly good compromise. Of course, if you can't afford to lose an hour's worth of data, and/or your server is very busy and causes the transaction log to grow quickly, then you may want to perform transaction logs more often, perhaps every 15 minutes or so. If you can't decide what transaction log backup interval to choose, I would err on having too many over having too few.

Of course, some organizations have policies that determine how much data they are willing to risk losing, and this will directly affect how often you perform transaction log backups. If your company doesn't have such a policy, then you might want to bring this issue to management, along with the pros and cons, and allow them to determine how much data they are willing to risk losing.

Configuring the Backup Database (Transaction Log) Task

Now that we have a little background on transaction logs, let's take a look at how to use the **Define Back Up Database (Transaction Log) Task** screen, shown in Figure 14.1.

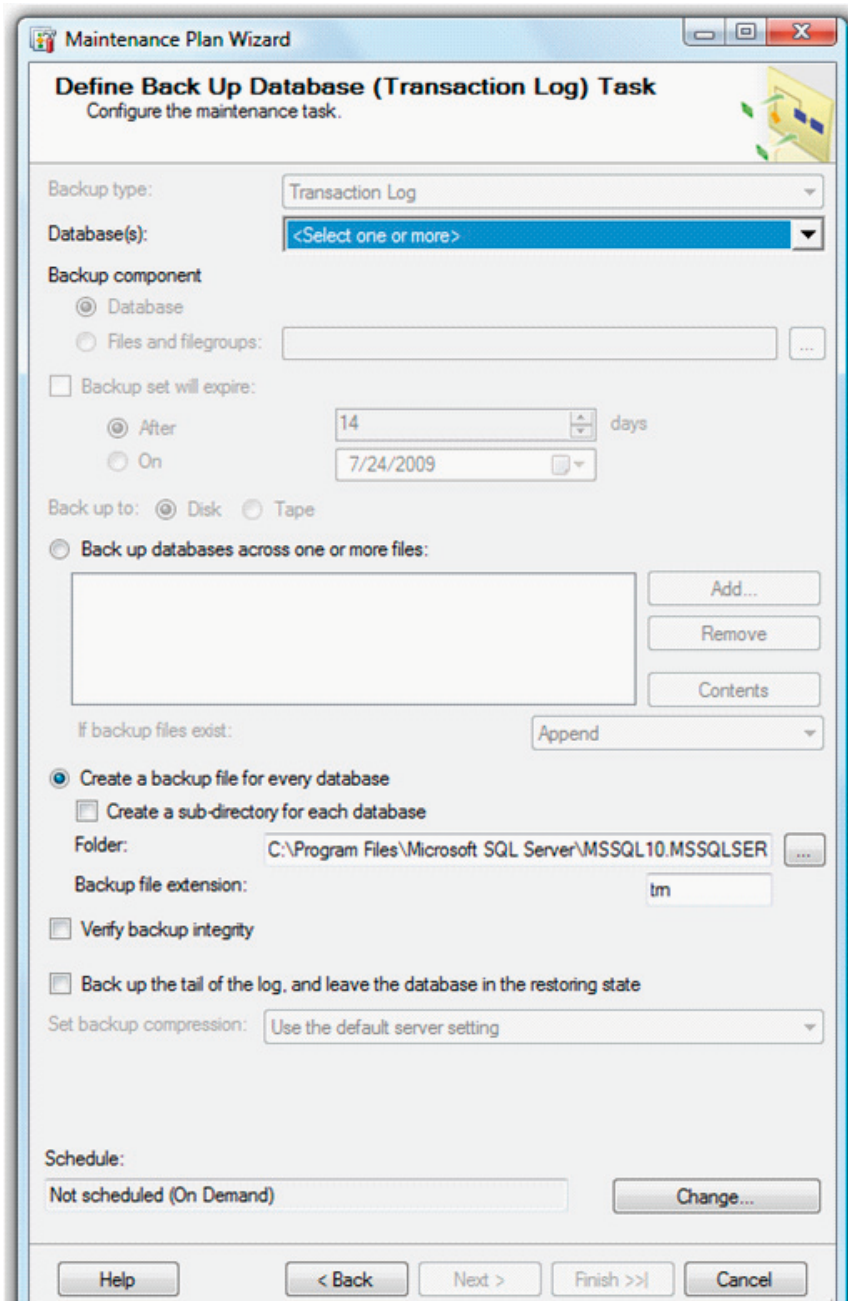

Figure 14.1: Configuring transaction log backups is almost identical to configuring full backups.

At the top of the screen, the **Backup type** option is set to **Transaction Log** and is grayed out. As you can see, after that, this screen of the wizard is virtually identical to the screen for full and differential backups, so it will only be described briefly in this chapter. Please refer to Chapter 12 for full details.

The first step, as always, is to select the databases on which you wish to perform transaction log backups. I highly recommend that you select the same databases in this screen as you did when you created the `Back Up Database (Full)` task. While it is possible to create two separate Maintenance Plans, one to do only full backups and one to do only transaction log backups, it would be a lot of extra work for no benefit.

One difference that you'll notice on the **Database Selection** screen is that the **System databases** option is grayed out, and that under **These databases** you will only see those databases that are using the full or bulk-load recovery models, since you can't perform transaction log backups on databases that use the simple recovery model.

> ***Use separate Maintenance Plans for system and user databases***
>
> *Most SQL Server instances are likely to have some user databases that use the full recovery model, alongside system databases, and other user databases, which use the simple recovery model. While you can create a single Maintenance Plan to cover all these databases, I prefer to create two separate Maintenance Plans, one for the system databases and one for the user databases, with each plan tailored to the specific maintenance needs of each set of databases.*

Under **Backup to** you will want to choose **disk**, and under **Create a backup file for every database** you should specify the same settings as you did for your full backups. This way, your full backups and log backups will be stored in the same location. In addition, leave the backup file extension to `TRN`, which is the default extension used for transaction log backups, and select **Verify backup integrity**, just as you should with all backups. If you have the Enterprise Edition of SQL Server, set the backup compression settings to the same as you did for your full backups.

Backing Up the Tail of the Log

One setting that is available on this screen, but is grayed out on the full backup and differential backup screens, is **Back up the tail of the log, and leave the database in the restoring state**. **Do not select this option**, as it won't work as part of a Maintenance Plan. The reason you see it is because this screen shares code with other parts of SSMS, and it is only from within SSMS that you might use this option to make a tail-log backup when recovering from a damaged database.

Creating the Job Schedule

The last step is setting the schedule for this task. The schedule screen is the same one we have seen many times before, so we don't have to examine it again. Just set the schedule so that transaction log backups occur on the schedule that you determine is best for your SQL Server environment. As I recommended earlier, I schedule transaction log backups at least once an hour, if not more often. You also need to keep in mind that you must perform at least one full backup of a database before you can perform a transaction log backup. Therefore, be sure that your full database backup schedule is set to start before your transaction log backup schedule.

If a scheduled transaction log backup is due to start while a full backup is occurring, then the transaction log backup job will wait until the full backup is done, and then it will run.

Summary

We have learned that the `Back Up Database (Transaction Log)` task is the second most important maintenance task that can be performed with the Maintenance Wizard, as it minimizes an organization's exposure to data loss, and truncates older data from a database's transaction log, which keeps your transaction log file to a reasonable size.

We are now done discussing the three backup maintenance tasks, and next, we will learn how to run the `Maintenance Cleanup` task, which can be used to help us remove old backup files.

Chapter 15: Maintenance Cleanup Task

This chapter completes our run through each of the tasks available through the Maintenance Plan Wizard and, perhaps fittingly, is where we clean up after ourselves.

If you run regular `Backup Database` tasks (full, differential, and transaction log) as part of the Maintenance Plans you create using the Maintenance Plan Wizard, then you may find that a lot of backup files are being stored on the local server, and this can very quickly eat up a lot of disk space.

In addition, if your Maintenance Plans write a report each time any maintenance task is executed, as I recommended in Chapter 3, then a text file will be created and stored each time. While these text files are generally small, many hundreds of them can be created each week and they can quickly take up a surprising amount of disk space.

Unless you regularly clear out older backup and text files, you could run out of disk space, bringing SQL Server to a grinding halt. Perhaps you have an `Execute SQL Server Agent Job` task that will warn you before you get to that point but, in any event, you must create some sort of plan to remove older files from your disk subsystem on a regular basis. This plan can be implemented using the `Maintenance Cleanup` task and this chapter will describe how the task works, and how to configure and schedule it using the Wizard.

Unfortunately, the Maintenance Plan Wizard implements the task in a somewhat compromised manner, allowing you to only remove **one** type of file (`BAK`, `TRN` or `TXT`) at a time, within a given Maintenance Plan. As such, we'll discuss ways around this limitation, including use of the Designer (see Chapter 16 onwards).

An Overview of the Maintenance Cleanup Task

An important part of every DBA's job is to determine a strategy for database backup storage. While you may be making backups to a local drive, this is not where you want them stored long term. If you store backups on the live server and the server suffers disk failure, then you could lose both the live databases and the backups. As such, it is important for the DBA to create a system whereby database backups are copied off the original SQL Server and stored in a safe location, preferably offsite. Ideally, you should be copying backups from your

SQL Server instances as often as transaction log backups are made. This way, should your server have a complete disk failure, you will have a full backup, along with the most recent transaction log backups, ready to be restored to a new server.

The `Maintenance Cleanup` task within the Maintenance Plan Wizard is designed to remove older files that have been safely copied to a separate offsite location, and which you no longer need to store locally. However, the task doesn't work quite as you might hope or expect. While it has the ability to delete older report text files (files with a `TXT` extension), full and differential backups files (files with a `BAK` extension), and transaction log backup files (files with a `TRN` extension), this task can only delete one of these file types at a time, from within a single Maintenance Plan created with the Maintenance Plan Wizard. Again, this limitation arises because the Wizard is unable to create multiple tasks within the same Maintenance Plan.

So, while ideally you'd like to clean up all these older files at once as part of a single Maintenance Plan, you can't, so you will need to find alternative ways to deal with this problem.

While there are several different options you can choose, here's what I recommend. First, within your central Maintenance Plan, use the `Maintenance Cleanup` task to delete the older report text files. Then, in order to remove the older `BAK` and `TRN` files, you have the options below.

- **Create two additional Maintenance Plans** using the Maintenance Plan Wizard, with one plan used exclusively for deleting older `BAK` files, and another plan used exclusively for deleting older `TRN` files. This requires you to create two additional Maintenance Plans, but it works well, especially if you are not familiar with T-SQL.

- **Use the Maintenance Plan Designer** to devise a single Maintenance Plan that will remove all three types of older files in one go. I explain how to do this, starting in Chapter 16.

- **Script a job to delete the older `BAK` and `TRN` files**, using some other tool, such as T-SQL, PowerShell, Command Prompt commands (like `delete`), or third-party programs that can be used to manage files on a file system. How to do this is beyond the scope of this book, but there are a lot of alternative ways of automatically deleting files from disk.

When you run the `Maintenance Cleanup` task, using its default settings, to delete Maintenance Plan report text files, it executes the following T-SQL code:

```
EXECUTE master.dbo.xp_delete_file 1,N'C:\Program Files\Microsoft
SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Log',N'txt',N'2009-07-
23T12:55:05'
```

As you can see, the T-SQL executes the `xp_delete_file` system stored procedure with a variety of parameters.

> ***The xp_delete_file system stored procedure…***
>
> *… can only delete Maintenance Plan report text files and native backup* `BAK` *and* `TRN` *files. If you want to delete other files, you will have to look for another option. Learn more about this system stored procedure in Books Online.*

The text report for this task looks as follows:

```
Microsoft(R) Server Maintenance Utility (Unicode) Version
10.0.2531
Report was generated on "HAWAII."
Maintenance Plan: User Databases Maintenance Plan
Duration: 00:00:00
Status: Succeeded.
Details:
Maintenance Cleanup Task (HAWAII)
Maintenance Cleanup on Local server connection
Cleanup Maintenance Plan report files
Age: Older than 4 Weeks
Task start: 2009-08-20T12:57:32.
Task end: 2009-08-20T12:57:32.
Success
Command: EXECUTE master.dbo.xp_delete_file 1,N''C:\Program
Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\
Log'',N''txt'',N''2009-07-23T12:57:32''
GO
```

It is a very simple report and is never longer than what you see here, because it can only delete one type of file at a time.

When and How Often to Clean Up Your Backup and Report Files

The `Maintenance Cleanup` job is generally quite lightweight, requiring few server resources, so you can schedule it to run most any time you want.

The more important decision to be made concerns how long you want to keep copies of these files on the local server, and so what constitutes "old."

If you create backups on a local drive, and then immediately copy them off onto another server or to tape, then you may not need to keep backups stored on the local server for very long. Generally, I like to keep one to three days of backups locally, but how many you can keep will depend on your available disk space.

If you are using report text files for maintenance job troubleshooting, they probably aren't very useful after a week, as new ones are being created all the time that are more current, and probably more useful when it comes to troubleshooting problems.

As the DBA, you will need to determine how long you should keep these files available on your SQL Server before deleting them.

Configuring the Maintenance Cleanup Task

Now that we have a better understanding of what the `Maintenance Cleanup` Task can and cannot do, let's take a look at the **Define Maintenance Cleanup Task** screen, shown in Figure 15.1.

Some options on this screen are a little misleading, so I need to walk you through it, one step at a time.

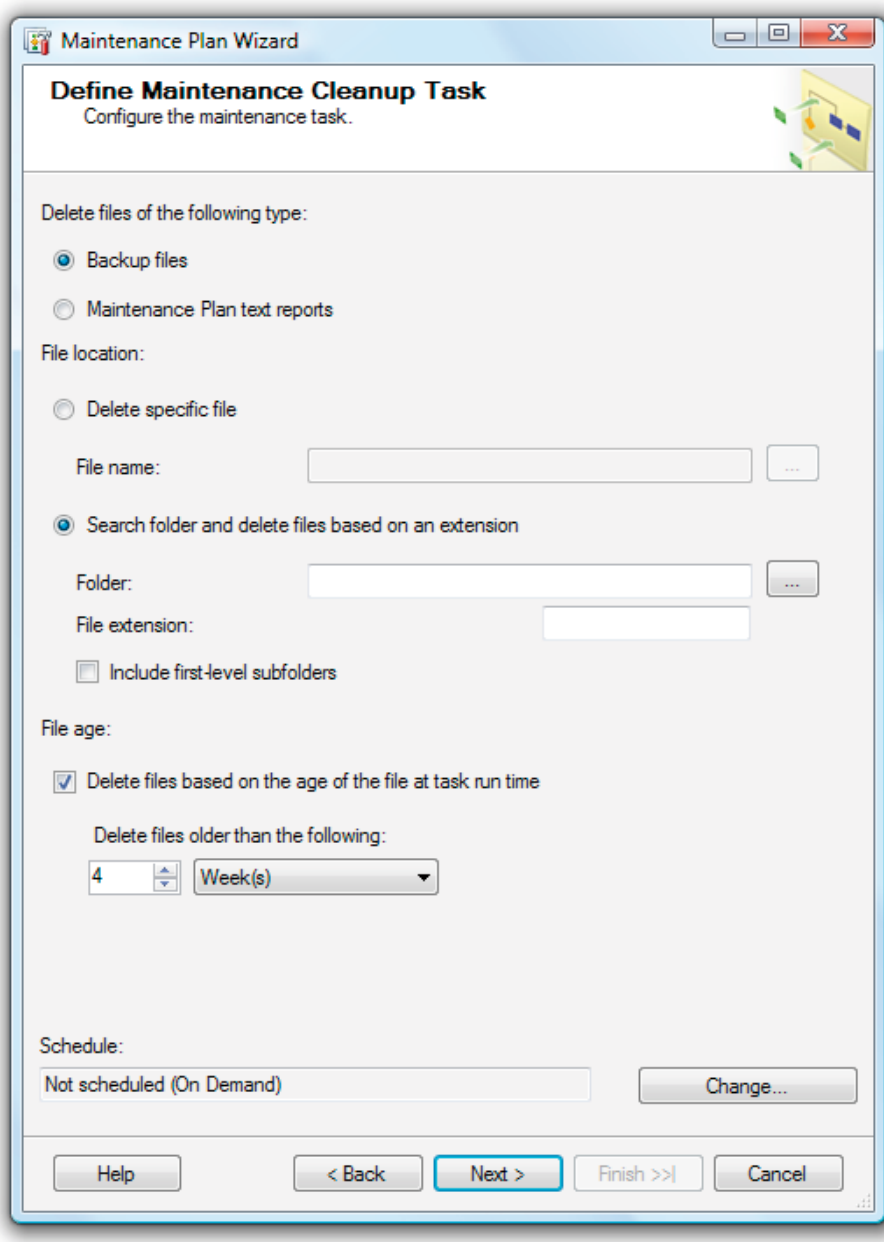

Figure 15.1: You can only delete one type of file at a time using the Maintenance Cleanup Task.

Specifying the type of file to delete

The first decision you have to make is which type of file to delete, as shown in Figure 15.2.

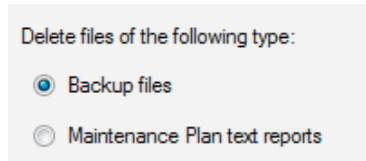

Figure 15.2: The names used here can be somewhat deceptive.

With the **Delete files of the following type** option, you have two choices.

- **Backup files**: This choice has a very deceptive name. It gives the impression that, if you select this option, you can delete both `BAK` and `TRN` files at the same time. Or at least, that is the impression I get from it. But this is not the case. What it really means is that, if you select this option, you can delete either `BAK` or `TRN` files, but not both. More on this in a moment.

- **Maintenance Plan text reports**: This option is more straightforward. If you select it, then you can delete Maintenance Plan text reports.

If this option were better designed, it would allow us to choose multiple options here, specifying either `BAK` files, `TRN` files, or `TXT` files in a single step but, unfortunately, that is not the way it is designed.

Specifying File Location

The next step is to specify the location of the file or, more commonly, files that you wish to remove, as shown in Figure 15.3.

Under **File location** you have two more choices. The first is to **Delete specific** file. This option allows you to select a single, specific file on disk, using the browse button to point to its location. I am not sure why you would choose this option, as virtually every `BAK`, `TRN`, and `TXT` file has a different name, so this option would only be useful if you wanted to delete the exact same filename each time this task runs. This would be a rare thing to want to do, but the option is available.

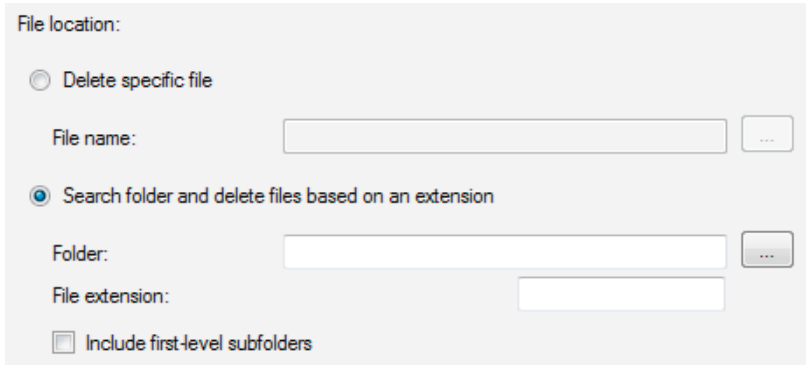

Figure 15.3: It would be very rare to only want to delete a single file.

The second option is to **Search folder and delete files based on an extension**. This is the default option, and the one you will use virtually all the time. It allows you to delete `BAK`, `TRN`, or `TXT` files based on the filename extension. This works well, because `BAK`, `TRN`, and `TXT` files almost always have unique names assigned to them, and using an extension is the easiest way to delete a lot of files in a single step.

When you choose this option, you also must provide two additional pieces of information. You must specify the folder where the files to be deleted are stored (you can use the browse button for this) and you must enter the file extension. If you chose **Backup files** earlier, the default value here is `BAK`. If you want to delete `TRN` files instead, then you will have to type in `TRN` manually. If you selected Maintenance Plan text reports earlier, the default value will be `TXT`.

You may also notice the option **Include first-level subfolders**. If you choose this option, not only will the extension type you entered here be deleted in the root folder specified by **Folder**, but so will any similar files in the first-level subfolders under this folder. This can be useful if you decide to use the **Create a sub-directory for each database** option when configuring the Database Backup tasks (see Chapter 12).

Delete files older than…

The last option is to specify the age of the files beyond which they will be removed, using the **Delete files based on the age of the file at task run time** option, as shown in Figure 15.4

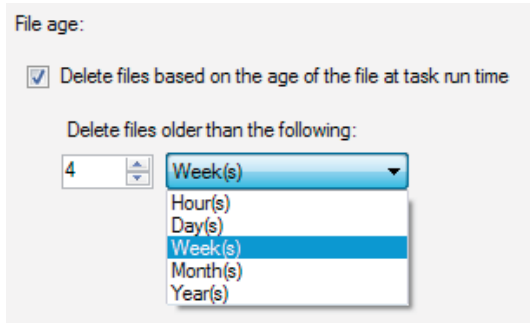

Figure 15.4: Don't uncheck this option.

This is another tricky option. First of all, don't deselect this option. If you do, the Maintenance Plan Wizard will assume that you want to delete all instances of the specified type of file, no matter how old they are. Of course, this is not what you will want to do. You will only want to delete files that are over a particular age.

Under **Delete files older than the following** you get to specify how long you want to keep the specified files. By selecting the number and time interval, you can choose a wide range of time frames, from hours to years. For example, if you choose 1 week, then all the files that are less than 1 week old are kept, and any files older than this are deleted.

As a general rule of thumb, I think you should keep one to three days' worth of backups (including full, differential, and transaction log) on your local server. I like to do this because, if I need to restore a database, most likely it will be from the most recent backup, and having it stored locally speeds up the restore process because the backup files are immediately available. This way, I don't have to find the files to restore, and copy them back on the local server to begin the restore process. This doesn't mean that I am not keeping additional offsite copies of these backups, because I am. I just find that if I need to do a quick database restore (assuming the hardware is fine), it is more convenient to perform the restore from a local backup than from a backup located offsite.

With regard to report text files, I generally delete any that are older than one week. In my experience, keeping them around longer doesn't benefit me when troubleshooting currently executing Maintenance Plans.

Creating the Job Schedule

And last, we have the **Schedule** option, which we have seen before, so I won't go into it in detail. When I create a task to delete report text files (which I keep for one week), I set the schedule so that the job runs once a week. If I create a task to delete backup files (both `BAK` and `TRN`), I usually run the job once a day.

These jobs take very few resources and you can schedule them to run most any time you want. Personally, I prefer to run this task after all my other tasks have run, in order to ensure that all the other tasks have run successfully before I delete any data. You never know when a job might fail, and you don't want to have deleted some data that might need later.

Summary

The `Maintenance Cleanup` task, while not very exciting, is nevertheless an important task, as you need to remove old backups and report text files, otherwise they clutter up your server and could lead to you running out of disk space. Unfortunately, this task is not well designed, and either requires you to create three separate Maintenance Plans to delete all three file types, or to use the Maintenance Plan Wizard, or other scripting options, to remove them.

We have now covered all the maintenance tasks available from the Maintenance Plan Wizard. In the next chapter, we will start to learn how to use the Maintenance Plan Designer.

Chapter 16: Introduction to the Maintenance Plan Designer

So far in this book, we have spent a lot of time learning how to use the Maintenance Plan Wizard to create a Maintenance Plan. We've investigated each of the eleven different maintenance tasks it can perform, and discussed how to configure and schedule each task, explaining the many settings that are available for each task.

Along the way, we discovered that much of the power of the Wizard stems from its simplicity. If you want to create a Maintenance Plan that runs a defined set of maintenance tasks the same way for a given set of databases, then the Wizard is a very powerful tool.

However, with its simplicity come certain limitations. One or two of the tasks, notably the `Maintenance Cleanup` task, can't be configured from within the Wizard in the way most DBAs need. In addition, there is no way to control what happens during the execution of a plan; for example, you can't include any logic within a plan to tell it what to do in case a particular task fails. The Wizard also provides a very limited scope for including "custom" maintenance tasks for a plan.

The Maintenance Plan Designer is a GUI-based tool that is built into SSMS and allows you to manually create Maintenance Plans, instead of using a wizard to step you through the process. It removes some of the limitations of the Wizard, allowing more flexibility over how you create your plans; for example, the ability to insert custom logic, define workflow, as well as providing access to additional tasks not available in the Wizard. The Designer also provides the only recommended route to modifying existing plans created using the Wizard.

This chapter provides a basic overview of the Maintenance Plan Designer, highlighting tasks and features that extend the capability of the Wizard, and including an introduction on how to use the Designer GUI.

In subsequent chapters, we'll discuss how to create Maintenance Plan tasks in the Designer (Chapter 17), how to make use of subplans and the conditional logic that the Designer provides, including the use of task precedence to control how Maintenance Plans operate (Chapter 18) and, finally, how to create complete Maintenance Plans in the Designer, and modify existing ones (Chapter 19).

Features Unique to the Maintenance Plan Designer

As noted in the introduction, the Designer adds a certain degree of flexibility, as well as additional features, to the creation of Maintenance Plans. As you might expect, with additional flexibility and power comes a steeper learning curve, and the Designer certainly takes a little longer to master than the Wizard.

However, having said that, many of the task configuration options are identical in both the Wizard and the Designer and so, by mastering the Wizard, you've already done a lot of the work necessary to master the Designer. In fact, having gained a little experience with the tool, many DBAs choose to create all their Maintenance Plans with the Maintenance Plan Designer, because of the greater flexibility it offers in creating Maintenance Plans.

The Maintenance Plan Designer can do everything the Maintenance Plan Wizard can, and offers the additional features below.

- **Create custom workflows and task hierarchies** – for example, the Designer allows you to:
    - **design and create multiple subplans**. In the Wizard, each task was automatically assigned to its own subplan, under the covers. The DBA had no control over this. In the Designer, you can design your own subplans, each of which can include various collections of tasks that run on similar schedules.
    - **establish and control precedence between tasks in a given subplan**. In this way, you can include conditional logic at certain steps in the execution of the plan, to control the next action to be performed, depending on the outcome of a previous task.
- Scheduling is done at the subplan level, not at the task level – when you added a task to a Maintenance Plan using the Maintenance Plan Wizard, you assigned a separate schedule to each task. Scheduling using the Designer is based on subplans, not tasks. The main difference is that a subplan can include two or more tasks and, because a schedule is assigned to a subplan and not a task, this means that a group of tasks may execute as a single entity. More on this later.
- **Execute a given task more than once within a single plan** – the Designer allows you to execute several different tasks of the same kind, as part of the same Maintenance Plan. The Maintenance Plan Wizard only allows a maintenance task to be included once in a Maintenance Plan.

- **Access to New Tasks** – the Designer includes two new tasks
  - `Execute T-SQL Statement` task – allows you to run any custom T-SQL code as part of your Maintenance Plan.
  - `Notify Operator` task – makes it easy for you to create a Maintenance Plan that will notify you if any problem occurs when a Maintenance Plan executes.

We will see examples of all of these features over the coming chapters. However, let's learn to walk before we run, and take a look at how to start up the Designer, then get a high-level feel for its GUI and the features that it provides.

Starting the Maintenance Plan Designer

Starting the Maintenance Plan Designer is as simple as starting the Maintenance Plan Wizard if not simpler, since it's the option I always find myself drawn towards, even when I intend to start the Wizard. From within SSMS Object Explorer, simply open up the **Management** folder, right-click on **Maintenance Plans** and then select **New Maintenance Plan...**, as shown in Figure 16.1.

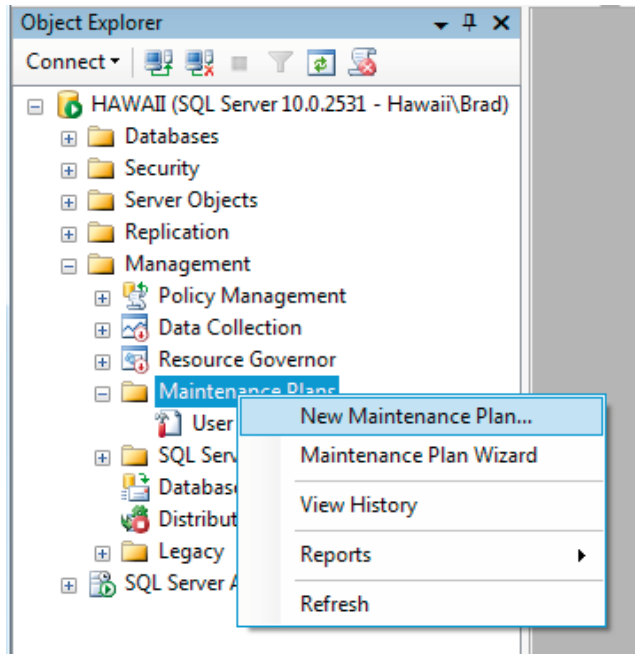

Figure 16.1: The New Maintenance Plan... option opens the Maintenance Plan Designer.

Having started the designer, the first job is to assign a name to your new Maintenance Plan, using the dialog box shown in Figure 16.2.

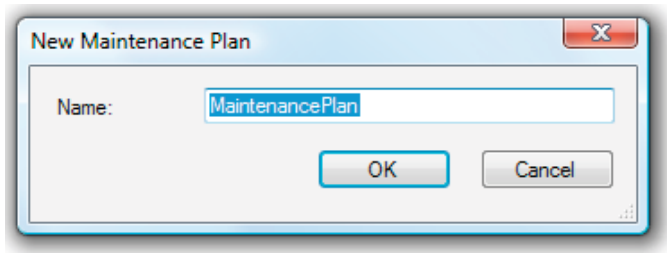

Figure 16.2: You must assign your new Maintenance Plan a name.

Pick a descriptive name that will help you and others to remember and understand the purpose of this plan. Once you've clicked **OK**, the Maintenance Plan Designer starts up within SSMS and you are ready to go.

Exploring the Maintenance Plan Designer

Having initiated the Designer, you are confronted with the Design surface, features and menu options shown in Figure 16.3.

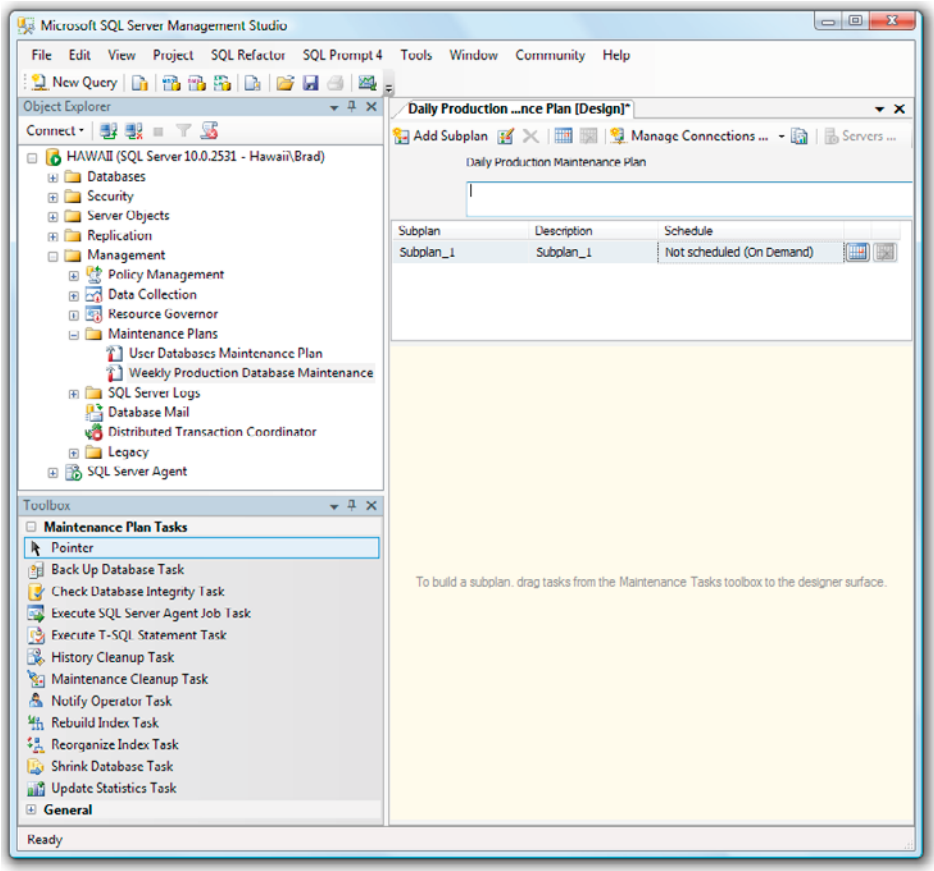

Figure 16.3: The Maintenance Plan screen is the GUI-based interface you use to create custom Database Maintenance Plans.

Before we drill into the specifics of using this tool, let's take a broad look at each major section of this screen.

Object Explorer

At the top left-hand side of the screen is the SSMS Object Explorer which, of course, is familiar to all users of SSMS. Other than being the point from which you can start the Designer, it plays no direct part in the creation of a new Maintenance Plan. The only other reason you might use the SSMS Object Explorer is to open an existing Maintenance Plan while creating a new one. To refer back to an existing plan, you can simply double-click on that plan's icon in Object Explorer and it will open up in Designer. The Maintenance Plan that you are working on will not close, but will be hidden beneath the one that just opened, and is accessible from a tab at the top of the window. In this way, you can have several Maintenance Plans open at the same time. To return to the original plan, simply select the appropriate tab.

While you will probably not be doing this often, it does allow you to check what you have done in a previous Maintenance Plan without having to close the Maintenance Plan that you are currently working on.

Maintenance Task Toolbox

At the bottom left-hand side of the screen, below the SSMS Object Explorer, is the Toolbox, shown in Figure 16.4, which is where all the available **Maintenance Plan Tasks** are displayed. As we will discuss later, you'll use the toolbox heavily when creating plans within the Designer, by dragging and dropping these tasks onto the design surface.

Most of the tasks in this Toolbox will be familiar to you, with the two previously-noted exceptions (the `Execute T-SQL Statement` and `Notify Operator` tasks, which we'll discuss in Chapter 17).

The Toolbox is divided into two sections, **Maintenance Plan Tasks** and **General**. All the tasks you need to use are available from the **Maintenance Plan Tasks** section. If the **General** section opens, I suggest you close it to prevent any potential confusion.

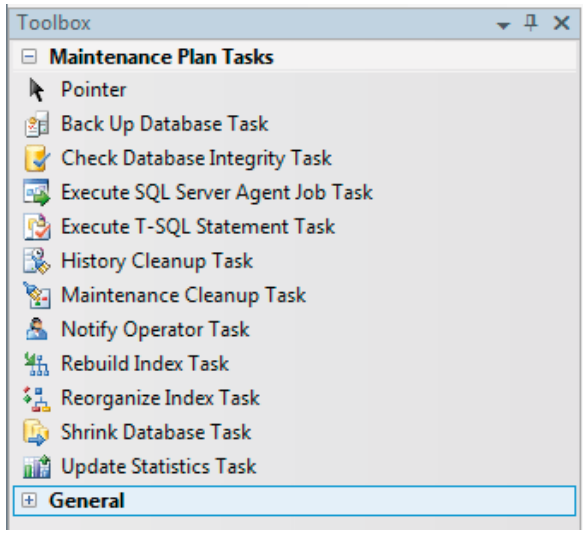

Figure 16.4: Focusing on the Maintenance Plan Tasks section of the Toolbox.

What's with the Pointer?

The very first item listed under **Maintenance Plan Tasks** *is called the* **Pointer**. *This is not a Maintenance Plan task, nor is it a useful feature when creating Maintenance Plans in the Designer. In theory, clicking the Pointer will put the "focus" back on the cursor within the design surface. If you place the cursor anywhere within the design surface and click, you will notice that the "focus" remains on the cursor, so the Pointer option is not needed. Why is it there? It's the same old story: the Designer was developed to be multipurpose within SSMS (for example, it is also used to create SSIS packages), and the Pointer is an artifact of this design. In other words, ignore it.*

Subplans and the Design Surface

The right-hand side of the Designer, shown in Figure 16.5, is dominated by the **design surface** (I've highlighted it with a border in the figure). It is on this surface that you design each of your Maintenance Plans, using the tasks provided in the Toolbox.

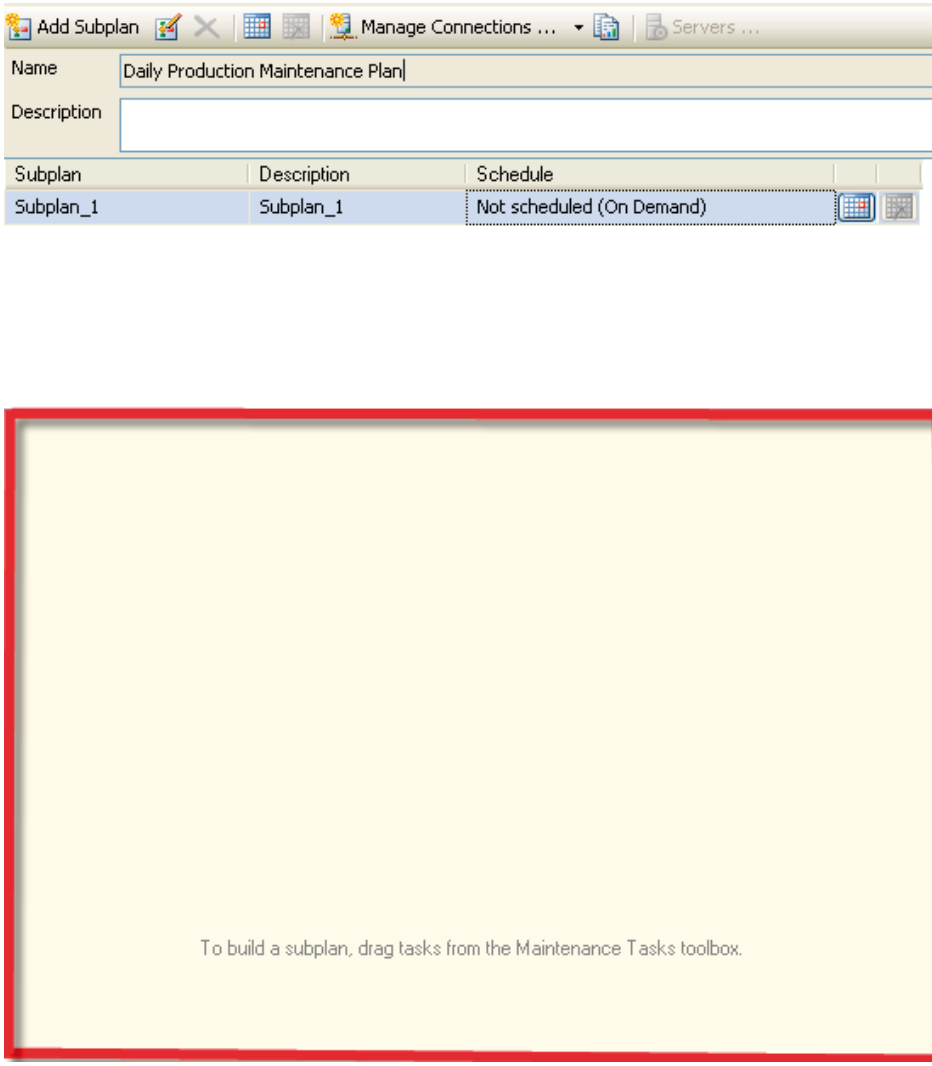

Figure 16.5: The right-hand side of the screen is dominated by the design surface (highlighted in red) where you visually create Maintenance Plans using the Maintenance Plan Designer.

Starting at the top of Figure 16.5, we see the Designer menu bar (covered later), and then a name textbox that is automatically filled in with the name you gave to the plan. Below that is a textbox where you can, optionally, enter a description of the plan, which I recommend you do. Below that is a grid describing the various subplans that comprise your Maintenance Plan. We'll take we brief look at this now, but will discuss subplans in a lot more detail in Chapter 18.

Subplans

The grid just above the design surface lists the subplans that comprise your Maintenance Plan, and allows you to set a schedule for each of them, as shown in Figure 16.6. This is different from how scheduling was done using the Database Maintenance Wizard. In the Wizard, each task had its own schedule. Using the Designer, scheduling is done by the subplan, and a subplan can contain one or more tasks.

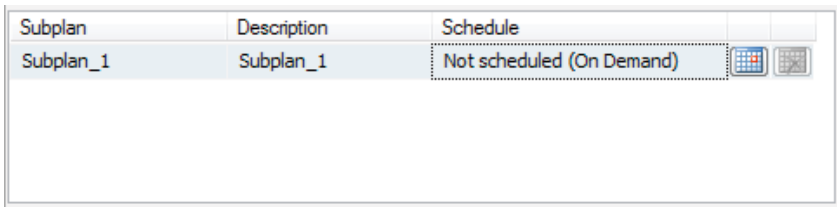

Figure 16.6: Subplans are created and managed here. Each subplan represents a collection of maintenance tasks that run on the same time schedule.

We will cover the topic of subplans in much more detail in Chapter 18 but, for the time being, you just need to know that a single Maintenance Plan can be made up of several subplans. Each subplan is made up of one or more maintenance tasks, and each subplan can be assigned its own time schedule. In other words, you can schedule the set of tasks defined by one subplan to run on a different schedule from the set of tasks in another subplan.

> ***Subplans and SQL Server Agent Jobs***
>
> *Behind the scenes, when you create a Maintenance Plan, a SQL Server Agent job is created for each subplan. As such, each subplan runs independently of other subplans within a given Maintenance Plan*

Certain tasks fit naturally onto the same subplan, whereas other tasks gravitate towards separate plans, as they tend to run on very different schedules. For example, the `Reorganize Index` and `Update Statistics` tasks would likely belong to the same subplan, occurring one after the other, whereas the task to perform daily full database backups would be on a separate subplan from the task to perform hourly transaction log backups.

When you first create a Maintenance Plan using the Maintenance Plan Designer, a default subplan, named `Subplan_1` is created, which is not scheduled. In order to schedule a subplan, simply click on the **Subplan Schedule** button on the right-hand side of the grid, to bring up the **Job Schedule Properties** screen. To the right of the **Subplan Schedule** button is the **Remove Schedule** icon (more on both these options a little later, when we discuss the Designer Menu bar, where they are also available).

If you were to create a new subplan by clicking on the **Add Subplan** icon on the top menu, then a second subplan, `Subplan_2`, would be displayed, as shown in Figure 16.7.

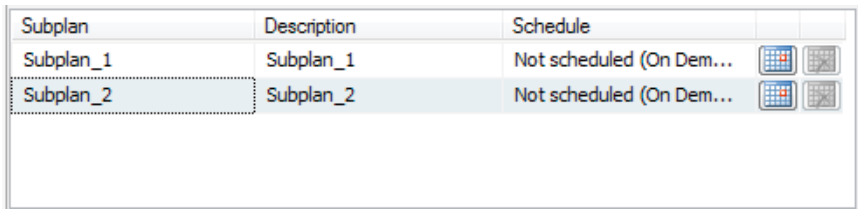

Figure 16.7: Each subplan is displayed in this window.

Each of these two subplans can have Maintenance Plan tasks associated with them (I'll show you how to do this next) along with their respective schedules. You can have as many subplans as you need, but I recommend that you keep them to a minimum, otherwise scheduling can get very confusing.

The Design Surface

The bottom section of the right-hand side of the screen is taken up by the design surface, which you can see, surrounded by a red border, in Figure 16.5. It is onto this surface that you can drag and drop tasks, in order to construct your plans. There is a separate design surface associated with each subplan. In other words, for every subplan you create, there will be a related design surface where you can assign Maintenance Plan tasks.

Dragging and Dropping Tasks

In order to demonstrate this, let's drag some tasks onto each of the Design surfaces for our two subplans. For the sake of demonstration, we'll add the `Check Database Integrity` task to `Subplan_1` and to `Subplan_2` we'll add the `Reorganize Index` task.

Click on the first subplan to highlight it. This activates the Design surface for that subplan. Drag the `Check Database Integrity` task from the toolbox and drop it on the design surface. The resulting screen should look as shown in Figure 16.8.

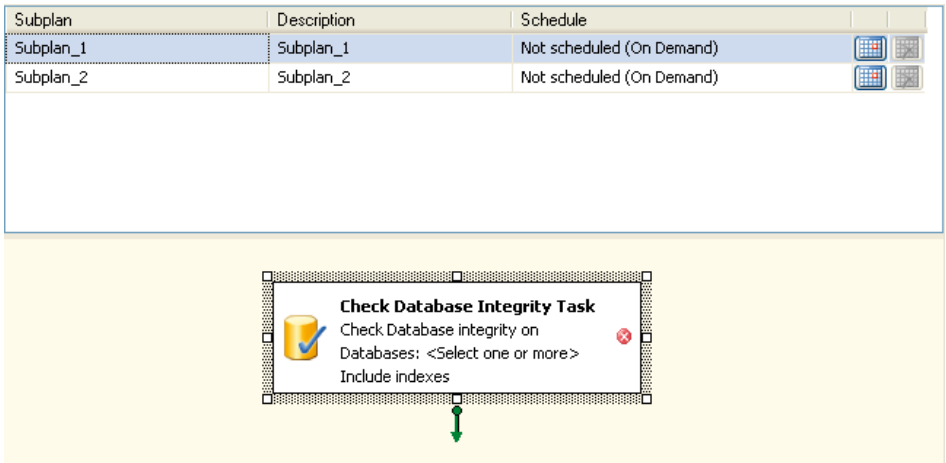

Figure 16.8: Subplan_1 has the Check Database Integrity task associated with its design surface.

Notice that `Subplan_1` is highlighted and the design surface below it includes the `Check Database Integrity` task. If we were to schedule this subplan, then the `Check Database Integrity` task would run on this schedule.

Next, repeat the process for the second subplan and the `Reorganize Index` task, as shown in Figure 16.9.

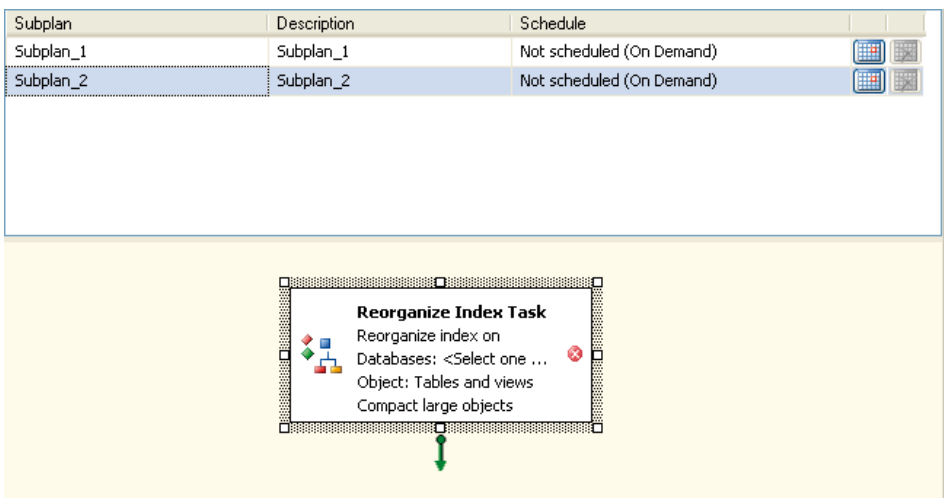

Figure 16.9: Subplan_2 has the Reorganize Index task associated with its design surface.

Notice that `Subplan_2` is highlighted and the design surface below it includes the `Reorganize Index` task. Now you can create a schedule just for this subplan, which will execute the `Reorganize Index` task, and any other tasks you decide to add to the subplan.

Task Configuration Boxes

We'll go through this process in a lot more detail in Chapter 18, when we examine each of the available maintenance tasks, but there are a few general points worth noting here in regard to the task maintenance boxes that appear on the design surface. When you select a given task, by clicking on its box, that task is in focus. A selected (in focus) task will be surrounded by a dotted gray box that can be expanded or contracted using one of the eight handles (small white squares). In fact, I expanded the task box shown in Figure 16.9 in order to make visible all the text inside it.

On the left-hand side of each task box is an icon that represents that task. I don't really find the icons obvious or memorable enough to differentiate each task, so I tend to just ignore them and rely on the task name, which is in bold type at the top of each task box. Below the task's name is a short description of the task, along with some configuration settings. This text will change from task to task, and depending how you configure the task.

The red circle with a white "x," to the right of the task boxes in Figures 16.8 and 16.9 indicates that, as of yet, neither of these tasks has been configured. Once a task is configured, this symbol will disappear. Of course, the absence of the symbol doesn't necessarily mean that that a task is configured *correctly*, only that it has been configured.

Finally, at the bottom of the task box is a green arrow, pointing down. This is used to link one task to another, to establish *precedence* between tasks, and insert conditional logic that controls the flow of tasks. In other words, you can use these arrows to specify the order in which tasks should execute within a given subplan and change the action of a dependent task based on the outcome of the precedent task. For example, if we were to drag an `Update Statistics` task onto the design surface for the subplan that also contains our `Reorganize Index` task, shown in Figure 16.9, then we'd want to use these green arrows to establish precedence between the two tasks. In other words, we'd want to specify, not only that the `Reorganize Index` task takes place before the `Update Statistics` task but also, potentially, that the latter should *only* be executed if the `Reorganize Index` task has been successfully completed. We'll discuss task precedence in a lot more detail in Chapter 18.

In case you start experimenting with the Designer before reading the rest of this book...

...be aware that when you add tasks to the same subplan, you should manually configure the precedence between tasks. If you don't, then all the tasks on the same subplan will try to execute at the same time, which, as you might imagine, can cause a lot of problems.

Designer Menu bar

At the top of the right-hand side of the screen, for each Maintenance Plan that you have open in the Designer, is a menu bar, as shown in Figure 16.10.

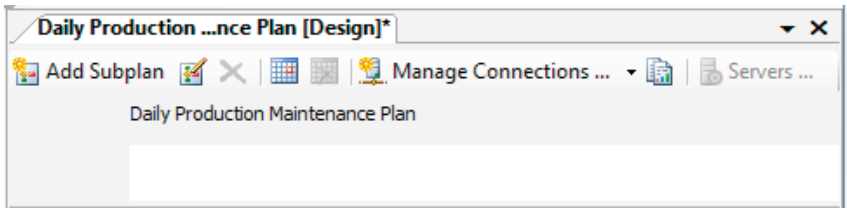

Figure 16.10: A lot of functionality is hidden in this small section of the screen.

This menu bar holds a surprising number of options, so let's explore each of the eight icons in turn. The first five options allow you to create (and remove) subplans and their schedules. The next two options pertain to the Maintenance Plan as a whole and allow you to configure the connections used by the plan, and the reports that are sent when the plan executes. The final option allows you to configure multiserver Maintenance Plans (an option I advise you to avoid).

Add Subplan

We've already used this icon in order to add a new subplan to a Maintenance Plan. When you click on the **Add Subplan** icon, the **Subplan Properties** screen appears where you can name, describe and schedule your new subplan, as shown in Figure 16.11.

Subplan Properties

When you click on this icon, you get the exact same screen shown in Figure 16.11. Why, you may ask? Basically, you define these properties for new plans using the **Add Subplan** icon, and use this **Subplan properties** icon to change the properties of existing subplans and their schedules.

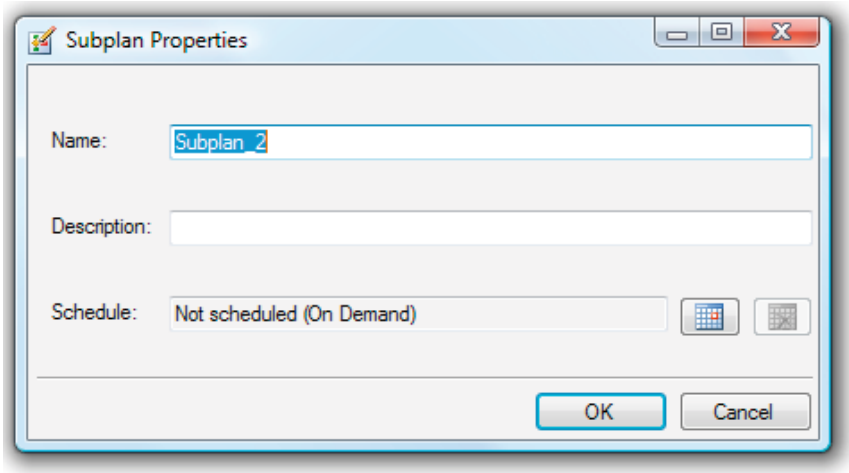

Figure 16.11: Adding a new subplan.

Delete Selected Subplan

The **Delete Selected Subplan** icon (which looks like an "X") is fairly self-explanatory, and will remove the selected subplan from the Maintenance Plan. When you first create a Maintenance Plan it will consist only of the default subplan, which cannot be deleted, so the option will be grayed out. When additional subplans are added, this option is activated.

Subplan Schedule

The **Subplan Schedule** icon looks like a calendar and is used to schedule a subplan. When you click on it, you are presented with the **Job Schedule Properties** screen that we have seen many times before in previous chapters, as shown in Figure 16.12.

Schedule a subplan (which may include one or more tasks) by highlighting it in the list, and clicking on this icon, or by clicking the equivalent icon in the subplan grid (Figure 16.7).

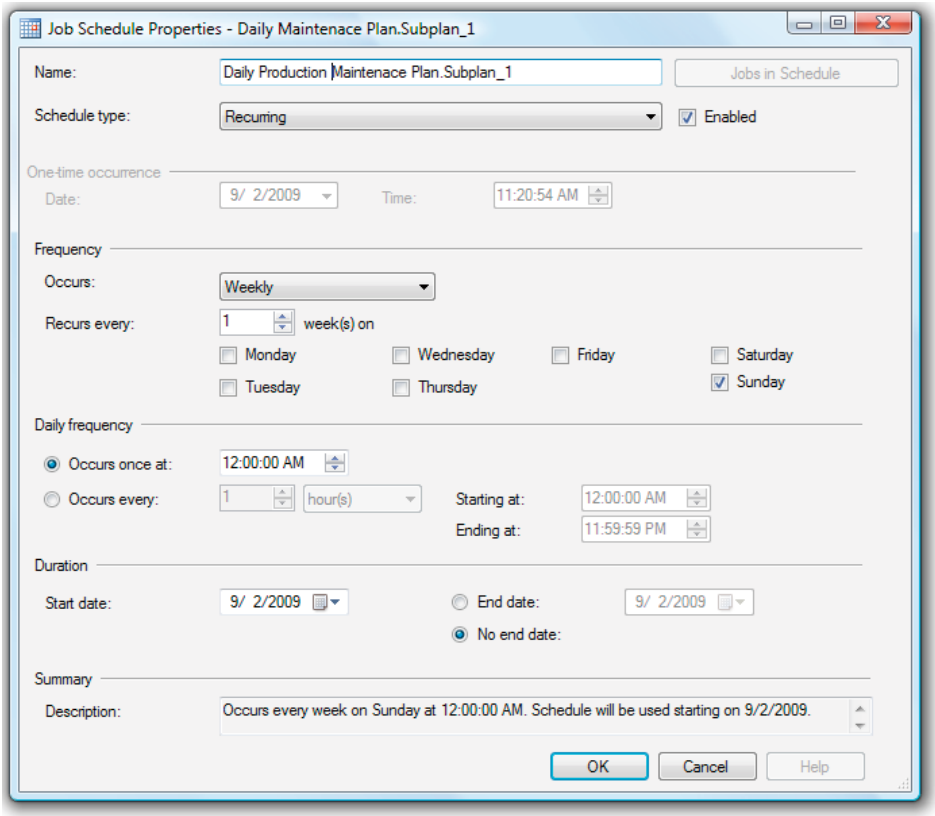

Figure 16.12: We saw this same screen when we learned about scheduling using the Maintenance Plan Wizard.

Remove Schedule

The **Remove Schedule** icon looks like a calendar that has been crossed out, and is used to delete the schedule for an existing subplan. Simply highlight any plan that has a schedule, and click the icon to remove the existing schedule for the selected subplan. Until you've created a schedule for at least one of your subplans, this option will be grayed out.

Manage Connections

When you create a Maintenance Plan using the Maintenance Plan Designer, the default assumption is that you want to create the Maintenance Plan on the local SQL Server (the SQL Server instance you are connected to via SSMS). In virtually every case, this is the correct

assumption, but the **Manage Connections...** option allows you to change the connection to a different SQL Server instance. When you click this icon, the **Manage Connections** screen appears, as shown in Figure 16.13.

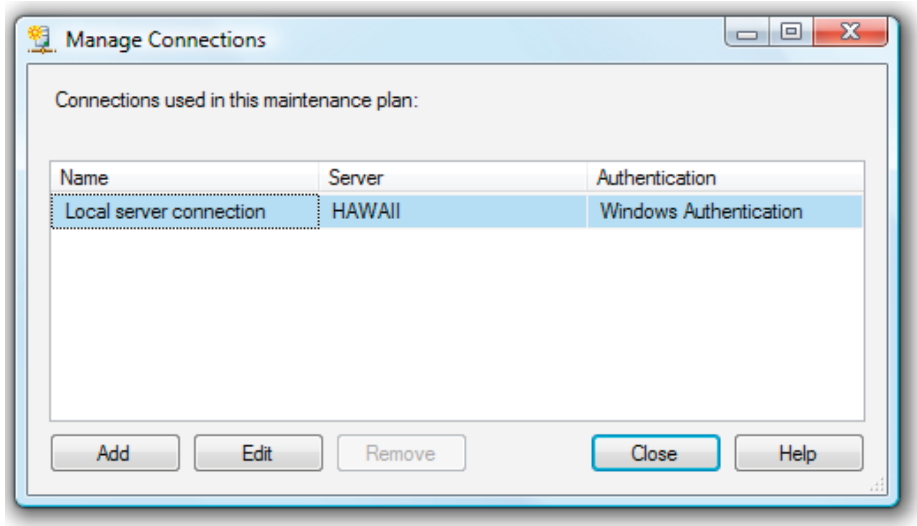

Figure 16.13: The Maintenance Plan Designer assumes that you want to create your Maintenance Plan on the SQL Server instance you selected from within SSMS.

On the **Manage Connections** screen, you see the connection information for the SQL Server instance you selected when you used SSMS to start the Maintenance Plan Designer. Should you want to change to a different SQL Server instance, or to change the authentication method, you could do so using the **Add** and **Edit** buttons. However, I suggest you keep everything simple and use the default, which is to connect to the local SQL Server instance. If you need the ability to connect to a different instance in order to create a Maintenance Plan there, it is better to do this using SSMS.

Reporting and Logging

The **Reporting and Logging** icon allows you to configure the sending of text reports that detail the tasks that were executed as part of the plan, and the level of detail that these reports provide.

When you click on the icon, it brings up the **Reporting and Logging** screen, shown in Figure 16.14. You may notice that it looks similar, though not identical, to the **Select Report Options** screen available from the Maintenance Plan Wizard.

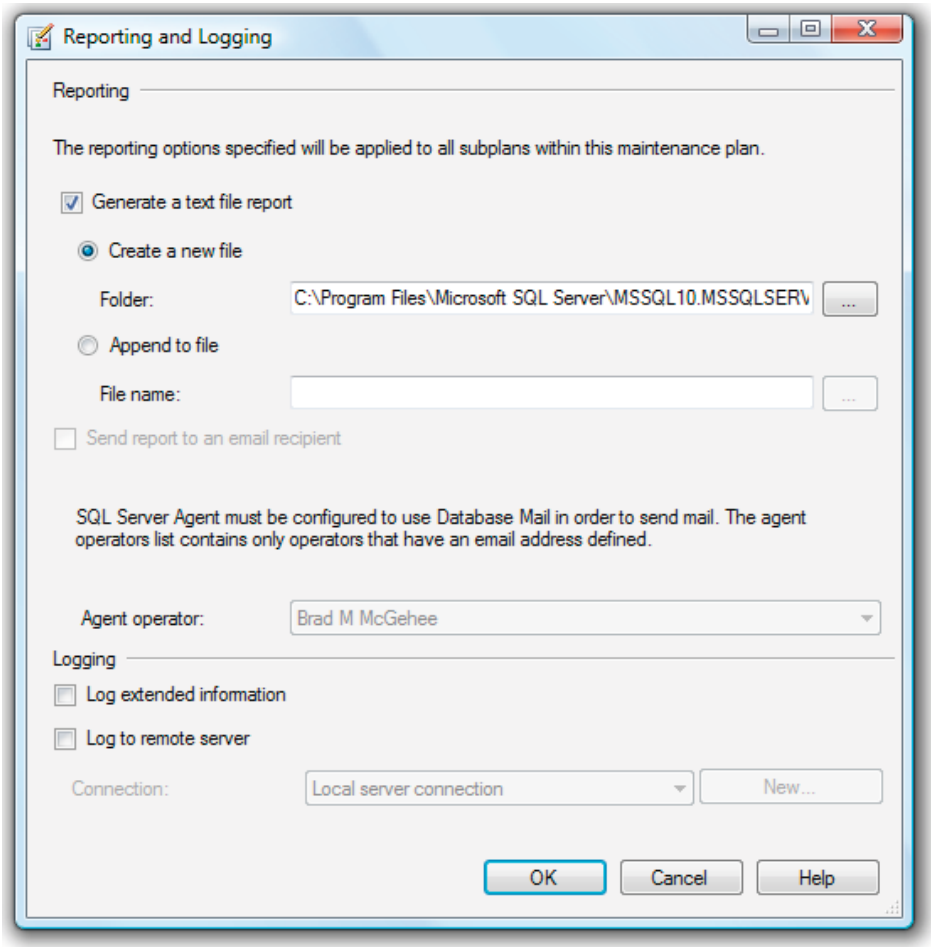

Figure 16.14: Use this option to configure the Maintenance Plan text file reports.

The first option is to **Generate a text file report**. It is selected by default and I strongly recommend that you leave it selected, as these reports are invaluable when troubleshooting a misbehaving Maintenance Plan. Notice that one report will be created for every subplan that is executed. So if a Maintenance Plan has four subplans, you'll get four reports each time that Maintenance Plan executes. This is different from how the Wizard worked, where one report was sent per Maintenance Plan.

If you decide to generate a text file report, you have two additional options.

- **Create a new file**. This is the default option and is identical to the option provided when configuring text file reports using the Maintenance Plan Wizard, where a new file is created each time a report is created. A default storage location for the report files is also provided.

- **Append to file**. This option forces all text file reports to be written to a single physical file. Obviously, this file will grow in size each time a Maintenance Plan job runs.

I suggest you stick to the default options, as separate files for each text file report makes it easier to find them, and the default location for the files is where DBAs always look to find them. If you change the location, then it will make it harder for other DBAs, not familiar with your configuration, to find these reports.

If you decide to ignore my advice, and select the single file option, then you'll also need to specify the path and the name of the file to which you want all text reports to be appended. Be warned though that it will make it much more difficult to find specific Maintenance Plan text file reports when they are all lumped together into a single file!

Immediately below the text file configuration options, in grayed out type, is the **Send report to an e-mail recipient** option. It is very similar to the option available from the Maintenance Plan Wizard option that sends an e-mail to an operator when a Maintenance Plan job completes. However, there is one important difference: *you can't use it*. Likewise, the next option on the screen, **Agent operator**, is grayed out and can't be used.

So how come you can set this option using the Maintenance Plan Wizard, but not with the Maintenance Plan Designer? It's not a mistake; this is intentional. The Maintenance Plan Designer has a special maintenance task called the `Notify Operator`, which is a much more powerful way of notifying DBAs if something goes wrong with a Maintenance Plan. We will discuss this task in Chapter 17.

So, if the `Notify Operator` task is used to send e-mail notifications, why is this option even available on this screen? Again, there is a valid reason. If you create a Maintenance Plan from scratch using the Designer then, yes, you must use the `Notify Operator` task for e-mail reports. However, say you created a plan through the Wizard, specifying that you want e-mails sent to a specific DBA, and then later needed to change this so that the mail was sent to a different DBA? In that case, if you opened the wizard-created plan in the Designer, you'd find that this **Send report to an e-mail recipient** option would be available and you could change the Agent who was to receive the e-mails.

Finally, on the **Reporting and Logging** screen, are the two **Logging** options, both exclusive to the Designer and unavailable in the Wizard.

The first option, **Log extended information**, is selected by default, and specifies that the text file report be as complete as possible. In fact, this is the same level of logging as that provided by the Maintenance Plan Wizard when it creates text log reports. If you deselect this option, you get a less detailed report. I recommend that you leave this option selected, as the additional information provides you with details that can make troubleshooting Maintenance Plans much easier. There is no downside to leaving this option on.

The second option, **Log to remote server**, allows you to send your text log reports to a different SQL Server than the one on which you are running the Maintenance Plan. I don't recommend this option because it adds complexity to your plans, but it can be used in cases where you want consolidate Maintenance Plan text file reports in a central location.

MultiServer Maintenance Plans

The eighth and final option on the Designer menu bar is called **Servers...** (it looks like a little server) and allows you to set up multiserver Maintenance Plans. The theory is that you can create a single Maintenance Plan on one SQL Server instance, and then run it on multiple SQL Server instances. The multiserver Maintenance Plan is created on what is called a master server, and then rolled out onto target servers.

Unfortunately, the theory does not translate well into practice in this case. This feature is awkward to configure, not very flexible, and is the cause of a lot of administrative headaches. As such, I have not discussed them in this book, and I don't recommend you use the feature. The **Servers...** icon is grayed out until master and target servers have been configured in SSMS. I suggest you don't do this. Leave the option grayed out and ignore it.

If you feel you have a need for such a feature, I suggest you investigate using T-SQL or PowerShell scripts which, in effect, can offer the same benefits, but are much more flexible and reliable.

Summary

Finally, we have covered the basics of how to use the Maintenance Plan Designer screen. Now it's time to begin learning how to configure individual maintenance tasks within the Designer.

Chapter 17: Configuring Maintenance Tasks Using the Designer

Having explored the Designer GUI in some detail in the previous chapter, we're now ready to investigate how to use the Designer to configure each of the eleven maintenance tasks that are available. You may be thinking that these are exactly the same eleven tasks we saw when using the Maintenance Plan Wizard, but that's not quite the case. Eight of the eleven tasks (`Check Database Integrity`, `Rebuild Index`, `Reorganize Index`, `Update Statistics`, `Shrink Database`, `History Cleanup`, and `Maintenance Cleanup`) are logistically more or less identical to the ones we configured through the Wizard. We'll cover most of these tasks in relatively little detail, avoiding as far as possible repetition of configuration options that are identical to what we saw when using the Wizard.

There is one task in the Maintenance Plan Designer, `Back Up Database`, which performs the same role as three separate backup tasks (Full, Differential, Transaction Log) in the Wizard. You may recall from Chapter 16 that one of the compelling advantages of the Designer is that, unlike the Wizard, it enables us to include multiple instances of the same task in a single Maintenance Plan. Therefore, rather than configure three separate backup tasks, as we did in the Wizard, in the Designer we simply configure three separate instances of the same task, one instance to do full backups, one differential, and one for transaction log backups.

This ability to include multiple executions of the same task in a single plan applies to any Maintenance Plan task within the Designer, but it is particularly useful in regard to the `Maintenance Cleanup` task, as it allows us to overcome the previously-noted limitations of Wizard version of the task, namely that it only allowed us to remove one of the three types of file (`BAK`, `TRN` and `TXT`) in any given plan. Using the Designer, we'll create a single plan that removes all three types of file in one go.

Finally, we'll cover in full detail two new tasks that are excusive to the Designer, namely `Execute T-SQL Statement` and `Notify Operator`.

A Note of Drag-and-Drop Caution

As we discussed in Chapter 16, adding tasks to a Maintenance Plan is as simple as dragging and dropping any of the tasks from the Toolbox directly onto the correct subplan's design surface.

Remember, though, that each subplan has its own design surface. If you are not careful, it is easy to assign a Maintenance task to the wrong subplan. Always double-check that the correct subplan is highlighted before dragging a task out of the toolbox. If you do make a mistake, it is easy to correct. Simply right-click on the misplaced task and select **Delete**, then highlight the correct surface and drag and drop the task. You cannot drag and drop tasks between subplans.

Check Database Integrity Task

As discussed in Chapter 5, the `Check Database Integrity` task is used to investigate the logical and physical integrity of all the objects in a database, looking for any corruption that could put data at risk. In order to configure this task in the Designer, the first step is to drag it into the design surface of the relevant subplan. Throughout this chapter, I'm going to keep things simple and use a single subplan and design surface, as in Figure 17.1.

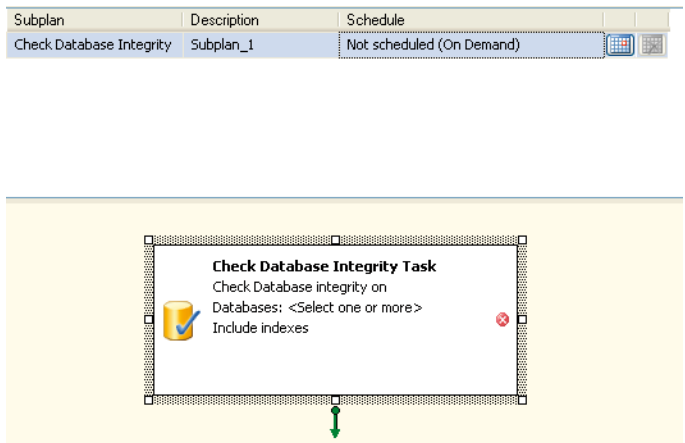

Figure 17.1: Maintenance Plan Tasks appear as rectangles on the design surface.

As mentioned in the previous chapter, the cross on the red background indicates that the task is not yet configured. To manually configure this and any other task in the Designer, right-click on it and select **Edit**, or double-click on the task, to bring up the task's configuration screen, shown in Figure 17.2.

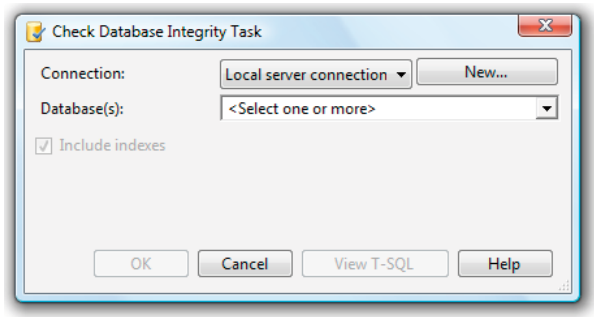

Figure 17.2: The configuration options screen for the Check Database Integrity Task.

The configuration screen for the `Check Database Integrity` task is similar, although not identical, to the one we saw in Chapter 5. The first thing you will notice is the **Connection** drop-down box, which is currently displaying **Local server connection**. Referring back to the **Manage Connections** option in the Designer menu bar, you'll recall that, by default, you'll create the Maintenance Plan on the local SQL Server (the SQL Server instance you are connected to via SSMS). Clicking on the **New...** button takes you to the screen where you can define a new custom connection (the same screen can be reached via **Manage Connections**). Unless you've previously created a custom connection, **Local server connection** will be the only available choice in the drop-down box, and is the one I recommend you stick to. I don't recommend using custom connections because they add complexity to your Maintenance Plans. If you need this capability, then you should probably be using custom T-SQL or PowerShell scripts instead.

> ***Scheduling tasks in Designer***
>
> *In the Wizard, the task scheduling option was at the bottom of each individual task configuration screen. In the Designer, scheduling is done at the subplan level, not at the individual task level, and will be covered in Chapter 18.*

Next on the screen is the **Database(s)** drop-down box. Clicking on **<Select one or more>** brings up the database selection screen shown in Figure 17.3 and with which we are by now very familiar.

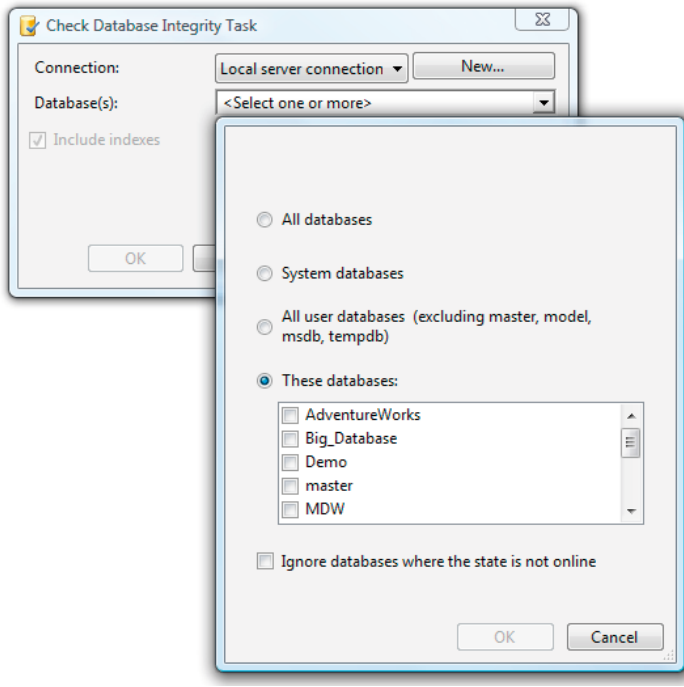

Figure 17.3: We have seen this database selection screen many times before.

For the purposes of this demo, select the `AdventureWorks` database, and then click on OK. The **Check Database Integrity Task** screen should now look as shown in Figure 17.4.

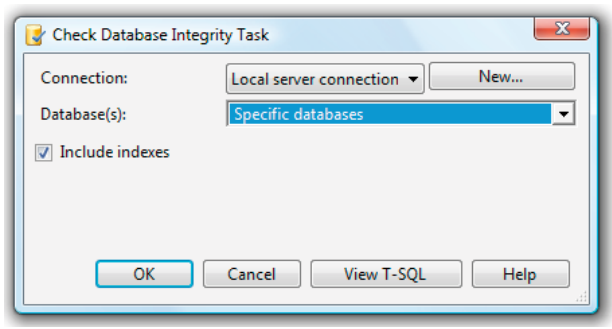

Figure 17.4: A specific database, AdventureWorks, has been selected, although we can't see the database name from this screen.

Now that a database selection has been made, the **Include indexes** option is available, which is checked by default, and means that both tables and indexes will be included in the check. As discussed in Chapter 5, this makes the task a little more resource-intensive but, in most cases, I recommend you leave it checked.

You have probably noticed a button on this screen that you have not seen before, namely the **View T-SQL** button. When you click on it, you get to see (but not modify) the T-SQL code that will be run when the task executes as currently configured (see Figure 17.5).

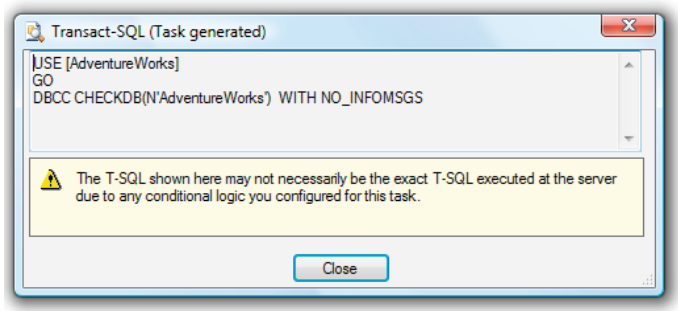

Figure 17.5: The Maintenance Plan Wizard allows you to view the T-SQL that will be run for each maintenance task.

If you're unfamiliar with T-SQL then this screen isn't much use to you, but most DBAs like to understand what T-SQL will be executed when they run a given task. Notice, however, the warning under the T-SQL code, indicating that the T-SQL you see may not be the exact T-SQL that is actually executed, due to the potential inclusion of further conditional logic that could alter the T-SQL that is executed, or mean that it is not executed at all. We'll cover this in more detail in Chapter 18, but suffice to say that the only way to really know what T-SQL was run for a particular task is to check the text file report that is created after a Maintenance Plan executes (see Chapter 3).

When you return to the design surface, you'll see that the `Check Database Integrity` task looks slightly different, as shown in Figure 17.6.

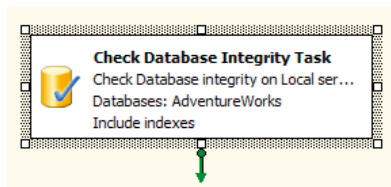

Figure 17.6: This task is configured and ready to execute.

First, the text underneath the name of the task now reflects our chosen configuration settings. Second, the red circle with the white "x" is also gone, indicating that this task has been configured and is ready to run.

This concludes our brief discussion of the `Check Database Integrity` task, and we still have ten more to go.

Rebuild Index Task

As discussed in Chapter 7, to which you should refer for full details, the `Rebuild Index` task will physically drop and rebuild any designated indexes, as a means to removing logical fragmentation and wasted space.

As always, the first step in configuring the `Rebuild Index` task in the Designer is to drag it from the Toolbox and drop it onto a design surface, at which point the **Rebuild Index Task** box appears and the task is ready to be configured, as shown in Figure 17.7.

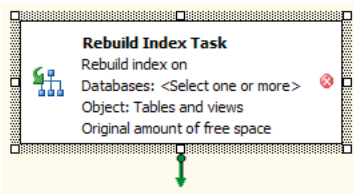

Figure 17.7: The Rebuild Index Task on the design surface.

To configure the `Rebuild Index` task, double-click on it (or right-click and select edit) to bring up its configuration screen, as shown in Figure 17.8.

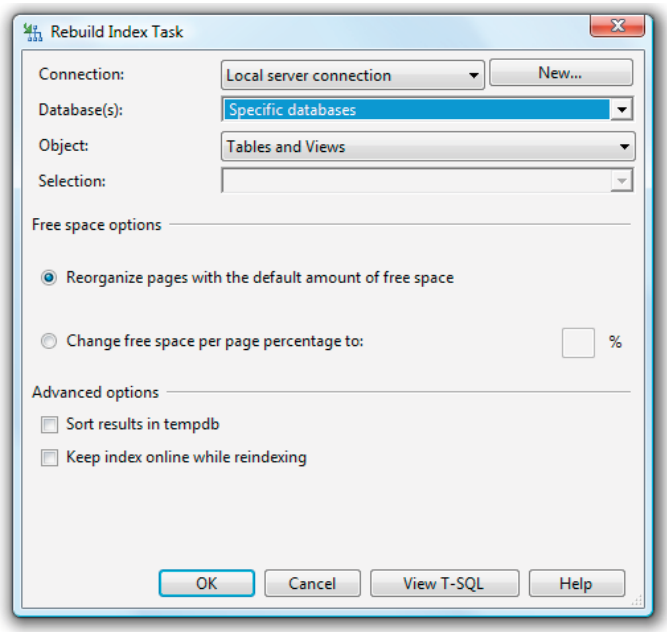

Figure 17.8: The options for the Rebuild Index Task for the Maintenance Plan Designer and the Maintenance Plan Wizard are identical.

In Figure 17.8, I've already selected a specific database to be the target of the `Rebuild Index` task, so that all other options are activated. Aside from the previously discussed **Connection** and **View T-SQL** buttons, these options are identical to those shown and described in the *Configuring the Rebuild Index Task* section of Chapter 7, and so will not be covered again here.

When you've configured the task as required, click **OK**, and the **Rebuild Index Task** box will reappear, displaying the specified configuration settings.

Reorganize Index Task

As discussed in Chapter 8, to which you should refer for full details, the `Reorganize Index` task is a much "gentler" version of the `Rebuild Index` task. It does not physically drop and rebuild the index but, instead, reduces logical fragmentation and minimizes unused space by reorganizing the leaf level pages.

Figure 17.9 shows the task box for the `Reorganize Index` task is its unconfigured state.

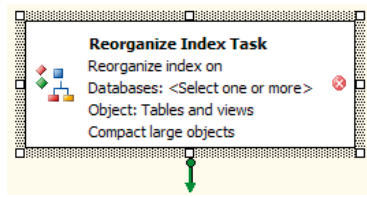

Figure 17.9: The Reorganize Index Task on the design surface.

Double-click on the task box to bring up the configuration screen, shown in Figure 17.10.

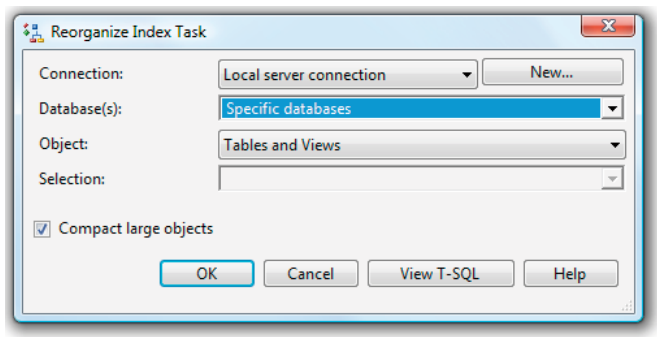

Figure 17.10: The Reorganize Index Task configuration options.

Again, these options are identical to those shown and described in the *Configuring the Reorganize Index Task* section of Chapter 8, and so will not be covered again here.

When you've configured the task as required, click **OK**, and the **Reorganize Index Task** box reappears, displaying the specified configuration settings.

Update Statistics Task

As described in Chapter 9, to which you should refer for full details, the `UPDATE STATISTICS` task causes the `UPDATE STATISTICS` command to be executed against all of the tables in the databases you select, ensuring that all index and column statistics are current, and so that the query optimizer has all the information it needs to determine the optimal execution plan for a given query.

Figure 17.11 shows the task box for the `Reorganize Index` task, as dropped onto a design surface in its unconfigured state.

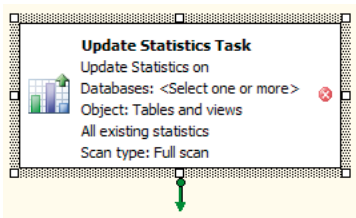

Figure 17.11: The Update Statistics Task on the design surface.

Double-click on the task box to bring up the configuration screen, shown in Figure 17.12.

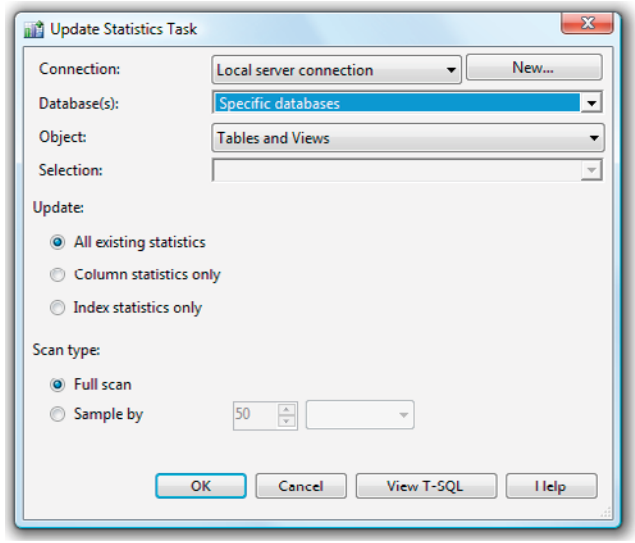

Figure 17.12: The Update Statistics Task configuration settings.

Once again, these options are identical to those shown and described in the *Configuring the Update Statistics Task* section of Chapter 9, and so will not be covered again here.

Shrink Database Task

In Chapter 6, I explained why I strongly advise you avoid using the `Shrink Database` task found in the Maintenance Plan Wizard. The exact same advice holds for the `Shrink Database` task in the Designer, so I won't be discussing it further here.

Execute SQL Server Agent Job Task

As detailed in Chapter 10, the `Execute SQL Server Agent Job` task allows you to run one (and only one) predefined SQL Server Agent job as part of a Maintenance Plan. The big advantage of using this task in the Designer is that, while the Maintenance Plan Wizard only allowed you to create one of these tasks per plan, the Designer allows you to create multiple instances of this task within the same plan, allowing you to include multiple jobs inside a plan.

Figure 17.13 shows the task box for the `Reorganize Index` task, as dropped onto a design surface in its unconfigured state.

Figure 17.13: The Execute SQL Server Agent Job Task on the design surface.

Double-click on the task box to bring up the configuration screen, shown in Figure 17.14.

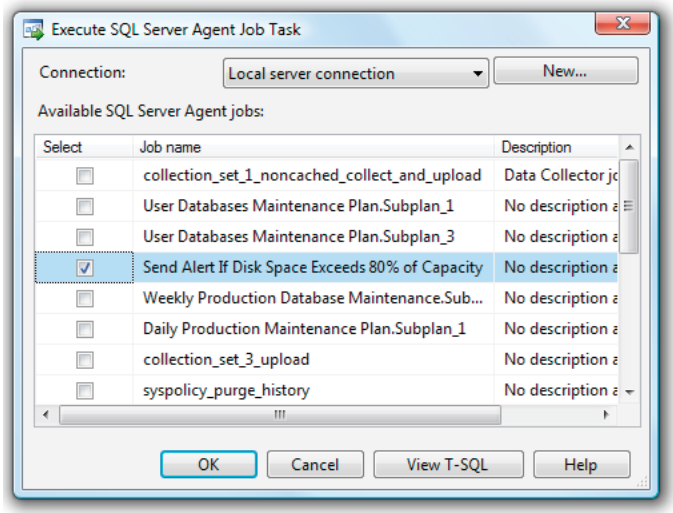

Figure 17.14: The Execute SQL Server Agent Job Task configuration screen. Your screen will look very different because your SQL Server instance will have different jobs than my SQL Server instance. Above, one job has been selected to run.

Yet again, these options are identical to those shown and described in the *Configuring the Execute SQL Server Agent Job Task* section of Chapter 10, and so will not be covered again here.

History Cleanup Task

As discussed in Chapter 11, the `History Cleanup` task simply removes old data from the `msdb` database, which the SQL Server Agent uses to store various bits of information about the jobs it runs. Figure 17.15 shows the task box for the `History Cleanup` task, as dropped onto a design surface.

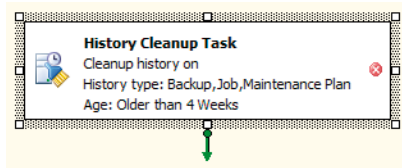

Figure 17.15: The History Cleanup Task on the design surface.

The configuration screen for this task is shown in Figure 17.16.

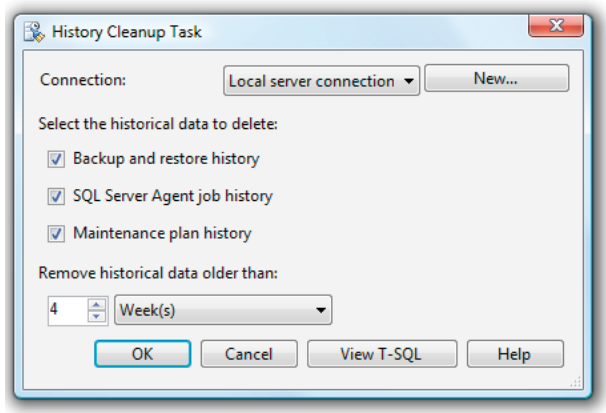

Figure 17.16: The History Cleanup Task configuration screen.

For the final time in this chapter, I need to note that these options are identical to those shown and described previously, in the *Configuring the History Cleanup Task* section of Chapter 11, and so will not be covered again here.

From here in, however, it gets a little more interesting. Of the four remaining Maintenance Plan tasks, the first two are available from the Maintenance Plan Wizard, but work slightly differently within the Maintenance Plan Designer, while the second two are new to the Maintenance Plan Designer.

Maintenance Cleanup Task

As discussed in Chapter 15, the `Maintenance Cleanup` task is designed to remove older backup (`BAK` and `TRN`) and text report (`TXT`) files that no longer need to be stored locally. However, in the context of the Wizard, this task had a very serious limitation: you could only remove one of the three types of file within any given Maintenance Plan. If you wanted to use the Maintenance Plan Wizard to delete all three types of files, you would have to suffer the

inconvenience, and added complexity, of creating three separate Maintenance Plans, one to remove each of the three types of files.

One of the compelling advantages of using the Maintenance Plan Designer is that it allows you to create a single Maintenance Plan that contains multiple instances of the same Maintenance Plan task. So, for example, we can add three different instances of the `Maintenance Cleanup` task to a single Maintenance Plan, and so delete all three types of older files in a single plan.

Let's take a look at how we can do this. Figure 17.17 shows the task box for the `Maintenance Cleanup` task, as dropped onto a design surface.

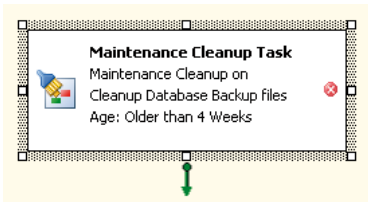

Figure 17.17: The Maintenance Cleanup Task on the design surface.

As you can see, by default the task is configured to clean up backup files that are more than four weeks old. We're going to want to execute three separate instances of this task, one to clean up old full and differential backup files (`BAK`), one to clean up old transaction log backup files (`TRN`) and one to clean up old text report (`TXT`) files. Therefore, the first step is to drag and drop two additional instances of the `Maintenance Cleanup` task to the design surface, as shown in Figure 17.18.

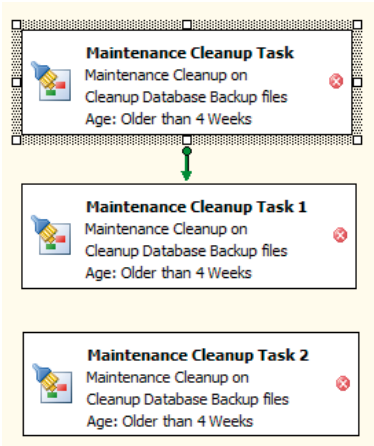

Figure 17.18: Multiple instances of Maintenance Cleanup Task on the design surface. While it may look like the green arrow is connecting the first two boxes, it is not.

In this example, I've dragged all three instances of the task onto the same design surface, so they are all part of the same subplan, and will execute according to the single schedule established for that subplan. If, for some reason, it was necessary to run one of the tasks on a different schedule, then it would need to be moved to a different subplan. We will take a deeper look at subplans in Chapter 18.

In Figure 17.18, while it may look as if the first two task instances are connected by a green arrow, they are not. If we wished to establish that these cleanup tasks should occur in a particular order, we would need to physically drag the arrow from the precedent task (at the start of the arrow) to any dependent task. Establishing the order in which tasks execute in a subplan is often very important because, if you don't, all the tasks in the subplan will try to run at the same time.

In this particular case, running all three of these tasks at the same time probably won't cause any problems, but in many other cases, running two or more tasks in the same subplan at the same time will cause problems, such as when you want to reorganize indexes and update statistics in the same subplan. The only way for this to work is to set the `Reorganize Index` task as the precedent task, which will execute first, with the `Update Statistics` task being a dependent task. However, in this example, we will not set any precedence, so we can forget about the arrows for the time being.

You'll notice, in Figure 17.18, that the Designer has assigned each instance of the task a subtly different name, in order to distinguish them. However, our first goal is to rename them in order to better identify what each instance is supposed to do. To change the name of a Maintenance Plan task, right-click on the instance, select **Rename**, enter the new name, and click **OK**. When you're finished the design surface should look similar to that shown in Figure 17.19.

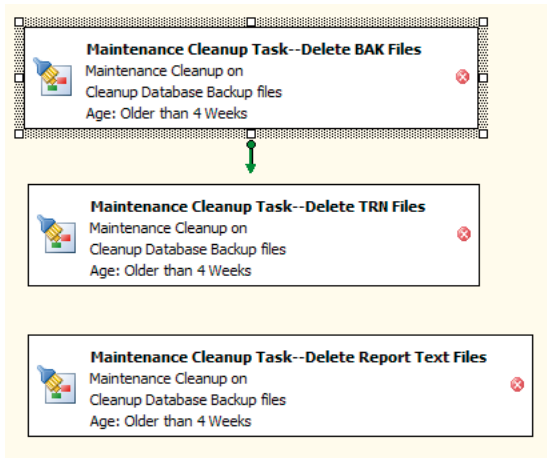

Figure 17.19: The names of Maintenance Plan Tasks can be changed.

As you can see, the names I have assigned to each instance make it clear exactly what each instance of the task is supposed to do. In addition, note that I had to increase the width of each box in order to prevent the new names being truncated, and so defeating the purpose of assigning more descriptive names.

Next, select the first task instance, which, in our example, is intended to clean up old `BAK` files, and bring up its configuration screen, as shown in Figure 17.20.

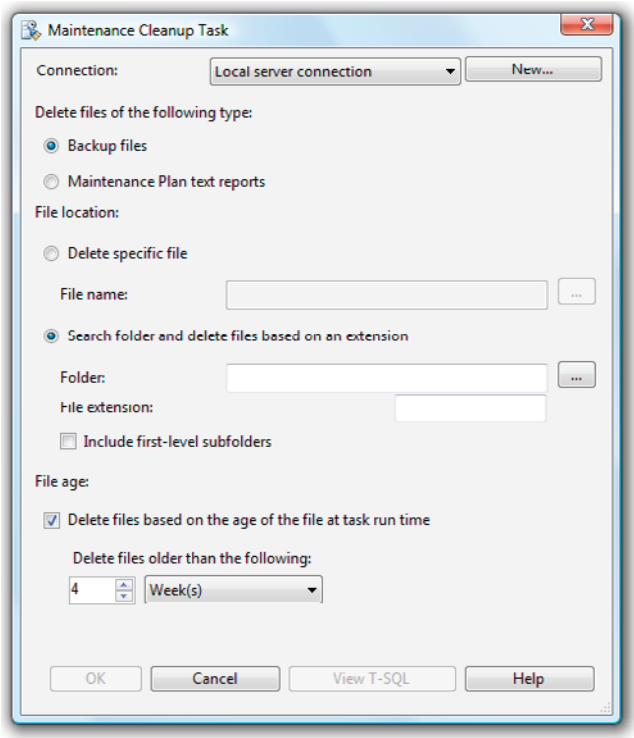

Figure 17.20: The Maintenance Cleanup Task configuration screen.

As you can see, the **Maintenance Cleanup Task** configuration screen is one we have seen before, in the Maintenance Plan Wizard, so I'll refer you to Chapter 15 for details of all the options. The goal is simply to configure each of the three `Maintenance Cleanup` task instances, one at a time, so that the first one deletes older `BAK` files, the second one deletes older `TRN` files, and the third one deletes older report `TXT` files. When you're done, you'll have a single plan that can perform all of the three required file cleanup operations.

Back Up Database Task

Using the Maintenance Plan Wizard, the process of backing up database files involved configuring up to three separate tasks:

- `Back Up Database (Full)` – a backup of all the data in a given database (see Chapter 12)
- `Back Up Database (Differential)` – a backup of any data that has changed since the last full backup (see Chapter 13)
- `Back Up Database (Transaction Log)` – a backup of the transaction log file (see Chapter 14)

You won't find any of these three options in the **Maintenance Plan Tasks** Toolbox. Instead, you will find a single backup task, `Back Up Database`, which you can use to perform full, differential, and transaction log backups. Figure 17.21 shows the task box for the `Backup Database` task, as dropped onto a design surface.

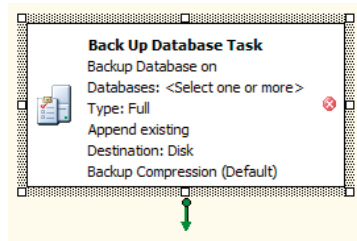

Figure 17.21: The Back Up Database Task handles full, differential, and transaction log backups.

The `Back Up Database` task configuration screen is shown in Figure 17.22.

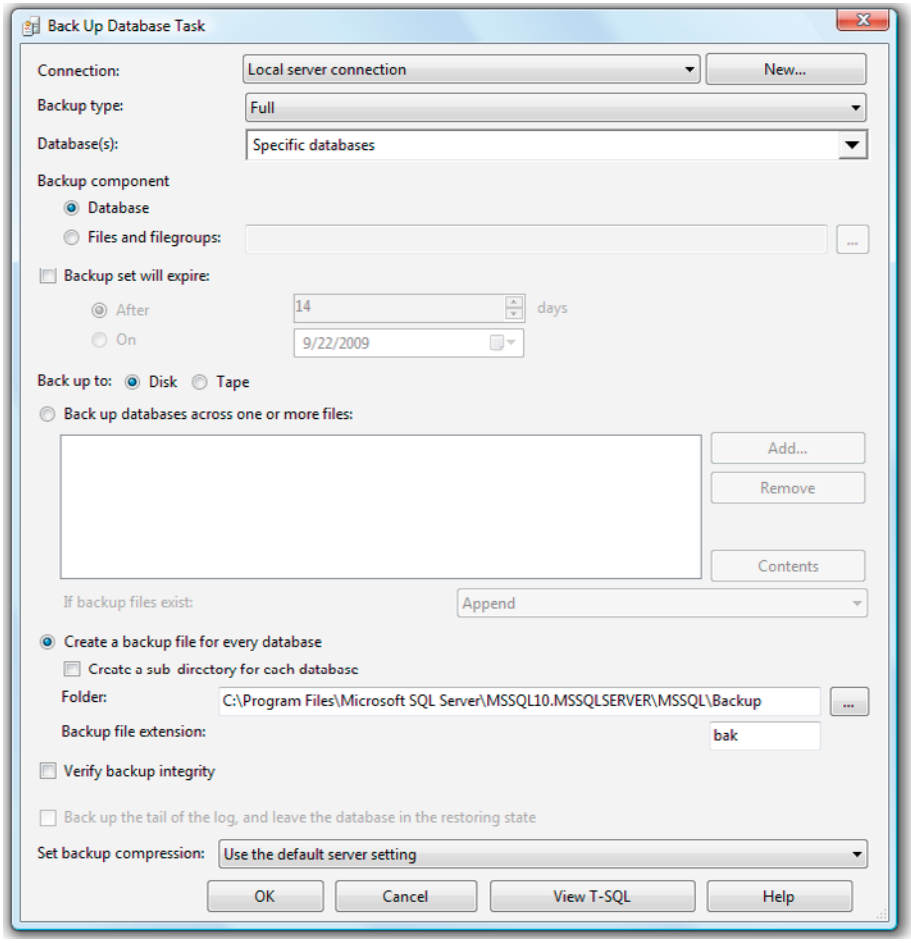

Figure 17.22: While we have seen this screen from the Maintenance Plan Wizard, one thing that is different is that now you can select a "Backup type" an option not available from the Wizard.

This screen is identical to the backup screen found in the Maintenance Plan Wizard, with one exception: the **Backup type** option is now available (it was grayed out in the Wizard), as shown in Figure 17.23.

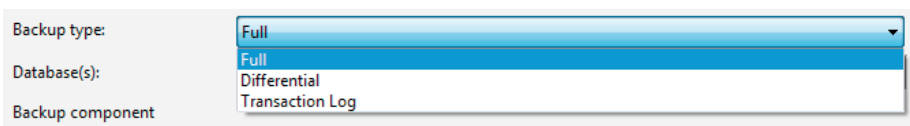

Figure 17.23: The Back Up Database Task offers the option for full, differential, and transaction log backups.

So, when creating a backup scheme for your SQL Server instances, using Designer, you simply define multiple instances of the `Back Up Database` task, one for each type of backup you need to make.

For example, if you needed to run full backups and transaction log backups on a given database (or set of databases), then you would create two instances of the `Back Up Database` task. Since these two tasks require very different schedules – for example, daily for full backups; but hourly, or even more frequently, for log backups – it is highly likely that you'll create each of the two task instances on a separate subplan, and assign it a separate schedule.

Other than selecting the **Backup type**, configuring the `Back Up Database` task is the same in the Designer as it is in the Wizard, and you can find full details of the available options in Chapter 12.

Execute T-SQL Statement Task

Up to this point, all the Maintenance Plan Tasks we have covered are available from the Maintenance Plan Wizard, albeit with some small changes in a few cases. Now, we take a look at the first of two Maintenance Plan tasks that are only available when using the Maintenance Plan Designer.

The `Execute T-SQL Statement` task allows you to run virtually any T-SQL code you want from within a Maintenance Plan. In many ways, it is similar to the `Execute SQL Server Agent Job` task in that it allows you to run custom T-SQL from within a Maintenance Plan. The main difference between these two is that the `Execute T-SQL Statement` task doesn't require that a separate SQL Server Agent job be created to run the T-SQL, as the T-SQL can be executed directly from this task.

Execute T-SQL Statement versus Execute SQL Server Agent Job

You might be thinking that if you can run the `Execute T-SQL Statement` *task, why would you need to run the* `Execute SQL Server Agent Job` *task? One key difference between these two options is that the* `Execute T-SQL Statement` *task only runs T-SQL code, while the* `Execute SQL Server Agent Job` *task not only runs T-SQL code, but it also run ActiveX, PowerShell, and operating system scripts.*

With the ability to execute virtually any T-SQL from within a Maintenance Plan comes a great deal of flexibility. However, in order to exploit this task sensibly, you'll need to ensure firstly, that the custom T-SQL performs a useful task that makes sense in the context of the overall Maintenance Plan and secondly, that the T-SQL code itself is crafted correctly.

If used sparingly, the `Execute T-SQL Statement` task can be useful. For example, let's say that you want to perform a database integrity check on a database, but you find the options available with the `Check Database Integrity` task too limiting. In this case, you could write your own T-SQL statement that executed the `DBCC CHECKDB` command, using the exact parameters that meet your needs.

If you decide to use the `Execute T-SQL Statement` task for such a purpose, then simply drag the task from the Toolbox, onto a design surface, as shown in Figure 17.24.

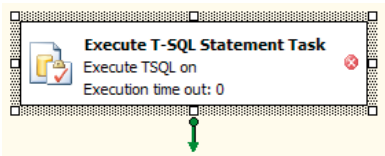

Figure 17.24: The Execute T-SQL Statement Task is very powerful, but it can also be overkill.

Configuring the task is easy; just add the required T-SQL code to the T-SQL statement box provided on the task configuration screen, shown in Figure 17.25.

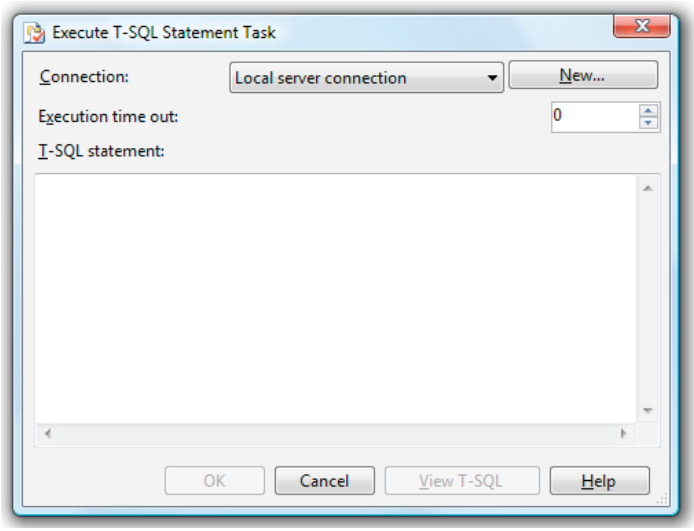

Figure 17.25: You can enter most any T-SQL command you want using the Execute T-SQL Statement task.

Of course, you will first want to create and test the T-SQL using SSMS, before including it in this maintenance task.

The **Execution time out** option on this screen is set to zero, which means that the T-SQL code you add can run as long as it takes to complete. If you want to prevent the code from taking an inordinate amount of time to run, you can set a value here, in seconds, which determines when the T-SQL code in this task times out, and the task is aborted. Of course, if your T-SQL code takes a long time to run, then it is probably not appropriate for use from within the `Execute T-SQL Statement` task.

Another point to notice about this task is that you can't specify a database, as you can with most of the other Maintenance Plan tasks. Because of this, this task is best suited to T-SQL code designed to run in any database, such as T-SQL code to run a custom form of the DBCC command, as described earlier. Or, if you are a clever T-SQL coder, you can write code that can check for existing databases and, based on some criteria you specify, perform specific actions.

You can write database-specific code if you want, but if you do, you should keep in mind that if you add or remove a database to a SQL Server instance you may break your code, requiring you to go back and modify the code to take into account the change in the databases on the instance.

Overall, if you have complex database maintenance needs that can only be satisfied using custom T-SQL then you may be better off creating your maintenance plans using custom T-SQL (or PowerShell) scripts in the first place. The Maintenance Plan Designer is intended to make database maintenance easier and faster; if you overcomplicate things by adding a lot of custom T-SQL tasks, then you are defeating its very purpose.

Notify Operator Task

The `Notify Operator` task is one of the main reasons why you may decide to create your Maintenance Plans using the Maintenance Plan Designer instead of the Maintenance Plan Wizard.

What this simple task does is to notify a designated operator (or several operators) when a Maintenance Plan task within your Maintenance Plan fails, succeeds, or completes. This adds a very useful level of granularity and control to the basic reporting options available in the Maintenance Plan Wizard. In the Wizard, a single screen is available, shown in Figure 17.26, which allows us to configure a text report to be sent to an individual operator (e.g. a DBA) whenever a Maintenance Plan completes execution.

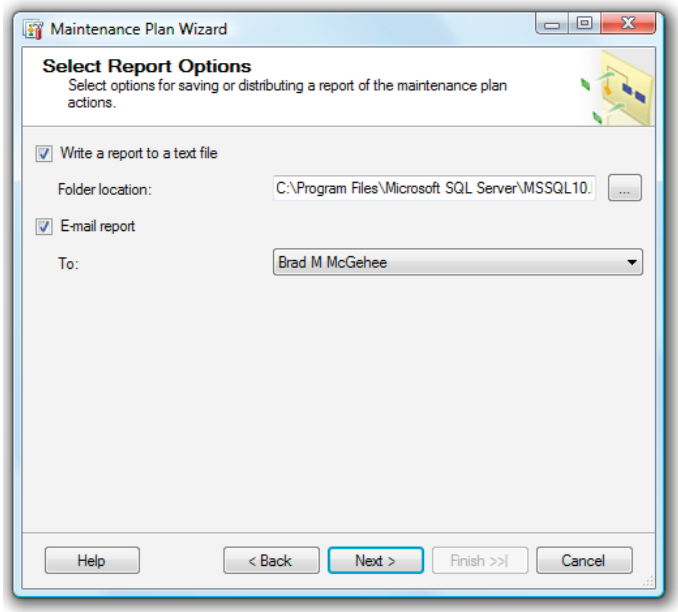

Figure 17.26: This screen, from the Maintenance Plan Wizard, is the only way that it can communicate with an operator.

While this is a great Wizard feature, it is also lacking in several important ways. First, it works at the plan level, so an e-mail is sent to the operator each time the plan is run. This contains details of all the tasks that were run as part of that plan. Just because an operator receives this e-mail doesn't mean that every step of the Maintenance Plan succeeded. If part of a Maintenance Plan were to fail, it's possible that the report will still be sent (assuming there is no major failure that prevents this), and the details of the failed step will be buried in the body of the report. In other words, you have to actually read the report text file to find out that something failed. Ask yourself: do I really want to read every e-mail sent to me by the Maintenance Plan Wizard to see if all the maintenance tasks inside it succeeded? The answer is probably "No."

Via precedence links, with the `Notify Operator` task in the Designer (more on this very shortly) you can associate the `Notify Operator` task with specific maintenance tasks in the Maintenance Plan, and so send e-mail reports to the designated operator on a task-by-task basis. Furthermore, you can specify more than one operator (without the use of e-mail groups). This adds a whole new level of flexibility and convenience. For example, let's say you have a Maintenance Plan that performs hourly transaction log backups. With the Wizard the poor, put-upon operator would receive an e-mail report every hour, which he or she would have to open and read to ensure the task completed successfully. With the Designer, you can configure the `Notify Operator` task to specify that the operator only receives an e-mail if the backup task fails, thus restricting e-mails to those occasions that require investigation and action.

Of course, you can also configure the `Notify Operator` task to send an e-mail when a job succeeds, or if a job completes (whether it fails or succeeds). In other words, you have lots of options.

> ***Using plan- and task-level reports***
>
> *Alongside reports of success or failure at the task level, you'll still want to have the plan-level reports written to file, which will detail the execution of every task within every subplan of your Maintenance Plan. See the Reporting and Logging section in Chapter 16.*

Furthermore, you can send the reports to multiple operators. By sending e-mails to individual operators (rather than to an e-mail group) you can exploit the fact that each operator can be configured to include their duty schedule. For example, operator A can receive and respond to failure reports when he or she is on duty, operator B when he or she is on duty, and so on.

> ***Creating and Configuring Operators***
>
> *Remember that operators are created and configured as a SQL Server Agent task, as described in Chapter 2.*

Now that you understand the potential of this task, let's take a closer look at it. When you first drag and drop it onto the design surface, it looks as shown in Figure 17.27.

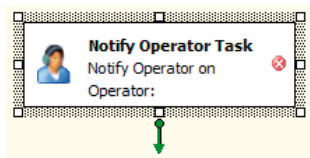

Figure 17.27: This simple task is very powerful.

The task is configured via the screen shown in Figure 17.28.

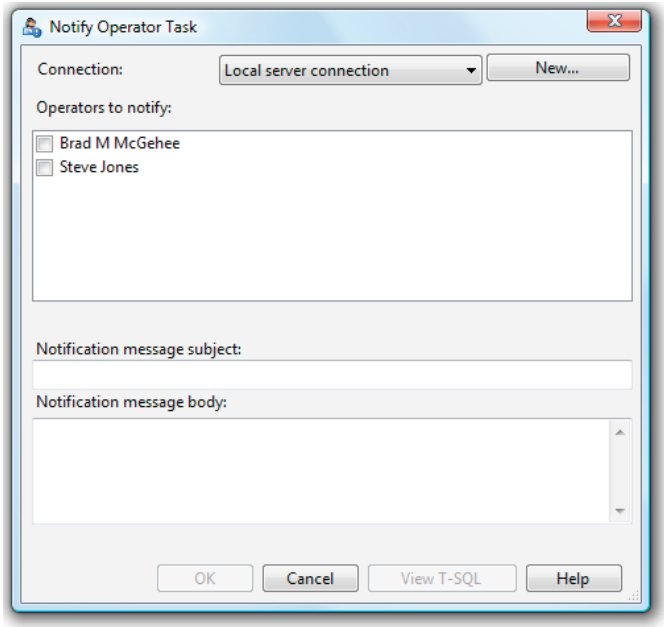

Figure 17.28: You can create custom e-mail messages to be sent to any selected operator based on the success, failure, or completion of any task within a Maintenance Plan.

On this screen you can create custom e-mail messages to send to the designated operators, by specifying the e-mail's **subject** and **body** text. So, for example, let's say you want to create a `Notify Operator` task that sends you an e-mail message if a full backup fails. In this case, you include text in the **Notification message subject** textbox and in the **Notification message body** textbox, which makes it abundantly clear that it's a message about a failed backup. This way, when you do get e-mail messages, you will know exactly why you received the message. The hard part is coming up with well-designed error messages for each separate task that can potentially fail. In the next chapter, we will see an example of how to create your own custom e-mail notifications.

Don't confuse the Notify Operator task with standard reporting and logging options

The `Notify Operator` *task only sends e-mail messages that you have created. It does not send text file reports. If you remember from the Reporting and Logging section in Chapter 16, text file reports are set up using the Reporting and Logging button at the top right-hand side of the Designer. In most cases, you will probably want to have the* `Notify Operator` *task send you e-mails about failed tasks and, in addition, you will want to have text file reports written to disk, just in case you need to follow up when troubleshooting failed tasks.*

Now, given the previous discussion, I bet you are wondering how to configure the `Notify Operator` task to send a specific e-mail when a particular Maintenance Plan task fails, succeeds, or completes? After all, these options don't seem to be visible on the configuration screen.

The secret lies in those green arrows that we've mentioned from time to time, and the subject of precedence. For example, let's say that if the `Back Up Database` task fails, you want an e-mail notification to be sent. The first step is to create the `Back Up Database` task. Next, you create a `Notify Operator` task. Finally, you link the two tasks together using the green arrows that determine the precedence between them. For example, if the `Back Up Database` task fails, the `Notify Operator` task will be executed, and the designated operator will then receive a customized e-mail alerting him or her to this fact.

Never fear, precedence will be made clear and demonstrated in the very next chapter.

Summary

As this point, you know the fundamentals of how to use the Maintenance Plan Designer and how to configure individual maintenance tasks. Now, you may be thinking, how do I make this all fit into a larger Maintenance Plan that includes many different tasks?

Before we can get there, there are two very important features that have been mentioned several times but need to be explained in full detail: **subplans** and **precedence**. The next chapter is dedicated to these topics. After that, in the final chapter, we will be ready to tie all this knowledge together and use it to create a full Maintenance Plan, from beginning to end, using the Designer.

Chapter 18: Subplans and Precedence

In the previous chapters, I have often referred to **subplans** and **precedence**, without giving much more than a cursory explanation of them. In this chapter, we will take an in-depth look at each of these important features of the Maintenance Plan Designer.

As noted previously, a single Maintenance Plan can be made up one or more **subplans**. Each subplan is made up of one or more maintenance tasks, and each subplan can be assigned its own schedule on which to run.

Precedence links can be used within a single subplan to control how the tasks within that subplan execute. Using these links we can dictate "what happens next" in a given subplan, based on the outcome of what is called "task branching."

Tasks on separate subplans execute independently of one another. In other words, you cannot modify the action of tasks in one subplan based on the outcome of tasks executed in another subplan.

As you devise your Maintenance Plans, you'll probably end up using both features; creating new subplans to accommodate tasks with conflicting schedule requirements, and using precedence links within a given subplan to exert control on the overall behavior of the Maintenance Plan. This chapter will show you how to use both techniques.

While it depends on the overall goal of your Maintenance Plans, I generally recommend using as few subplans as you can, restricting their use to those tasks that really do need to run at different times. Instead of relying on multiple subplans for your Maintenance Plans, I want to suggest that you focus your efforts instead on the power of precedence to control how your Maintenance Plans execute.

Subplans

When you create a Maintenance Plan using the Maintenance Plan Designer, you can either lump all the Maintenance Plan tasks into a single subplan that runs according to a single, specific, schedule, or you can spread your Maintenance Plan tasks over one or more subplans, each with its own schedule. Let's look at the pros and cons of each option.

Using a Single Subplan: Pros and Cons

Using a single subplan keeps your Maintenance Plan simple and straightforward. Every task is on a single design surface, and easy to see.

If cleverly designed, you can use precedence (more on this topic coming up very soon) to determine if and when a particular task executes, and how it affects the execution of another task. For example, you could have a subplan that contains a `Check Database Integrity` task, a `Back Up Database` task and a `Notify Operator` task. You can set up precedence rules that dictate that, if the integrity check succeeds, then the backup task should be run immediately, but if the integrity check fails, an e-mail notification should, instead, be sent to the operator, letting him know that the `Check Database Integrity` task has failed.

Some Maintenance Plan tasks fit together nicely on the same subplan and on the same schedule. For example, every task that is performed only once a week, during a scheduled maintenance window, is likely to be on the same subplan.

Other tasks have incompatible schedule requirements and so are better placed on multiple subplans, which run on different schedules. For example, a task that performs full backups once a day can't be included on the same subplan as a task that performs transaction log backups once an hour.

Using Multiple Subplans: Pros and Cons

Multiple subplans allow you to design a Maintenance Plan with multiple schedules so that specific tasks can execute independently, on a schedule of your choice. In fact, the need to include multiple schedules within a single Maintenance Plan is the only real reason to use multiple subplans. If particular tasks have very different schedule requirements, then you simply place them in different subplans with an appropriate schedule in each case. For example, if you had certain tasks that needed to be run hourly, others daily, others weekly, and others monthly, then you'd probably need four separate subplans.

If you design your Maintenance Plans cleverly, using multiple subplans, you may be able to keep the number of Maintenance Plans you need to create for a single server to as few as one.

On the downside, if you use multiple subplans, you can only see one design surface at a time from within the Maintenance Plan Designer. If each design surface has one or more Maintenance Plan tasks, you can't view them as a whole. This makes it somewhat more difficult to get a big picture of what the overall plan is doing, and to troubleshoot potential problems.

Precedence only works for tasks within a single subplan; it cannot cross subplans. This means that you can't make the action of a task in one subplan dependent on the behavior of a task in a separate subplan.

In almost all cases, my advice is as follows:

- tasks that are not schedule-dependent, but are precedence-dependent, should be on the same subplan
- tasks that are schedule-dependent, and not precedence-dependent, should be on separate subplans, each with their own schedule.

Using Subplans

In order to demonstrate how to use subplans, and to set their schedules, let's look at a simple example. Let's assume that we want to create a Maintenance Plan that does a full backup once a day and a transaction log backup once an hour, and that we want to put each in its own subplan.

Let's start the Maintenance Plan Designer afresh so that we have only the default subplan and no tasks currently added to its design surface, as shown in Figure 18.1.

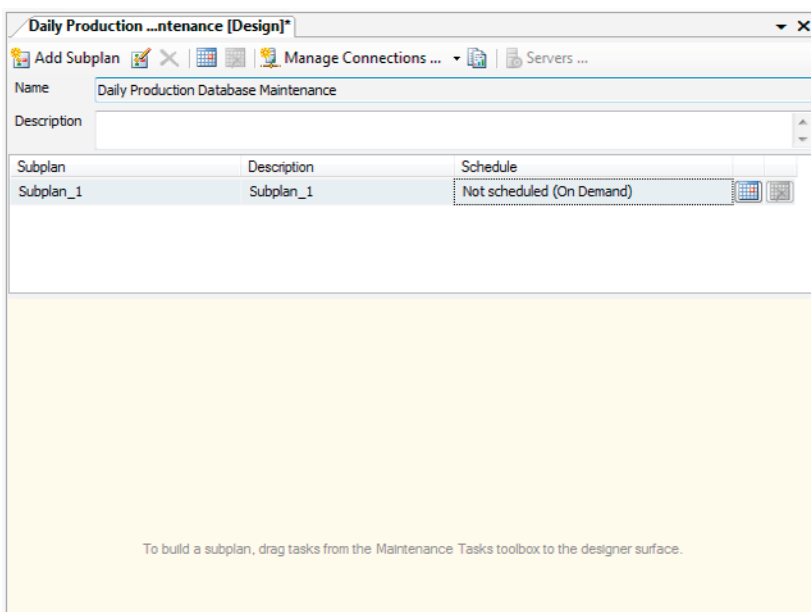

Figure 18.1: Let's start out with a new Maintenance Plan.

The first step is to add the `Back Up Database` task to the default subplan, and then configure it to perform full backups on the required databases. In this case, we'll perform a full backup on the `AdventureWorks` database. Once the task has been dragged and dropped onto the design surface of **Subplan_1**, and configured, it will look as shown in Figure 18.2.

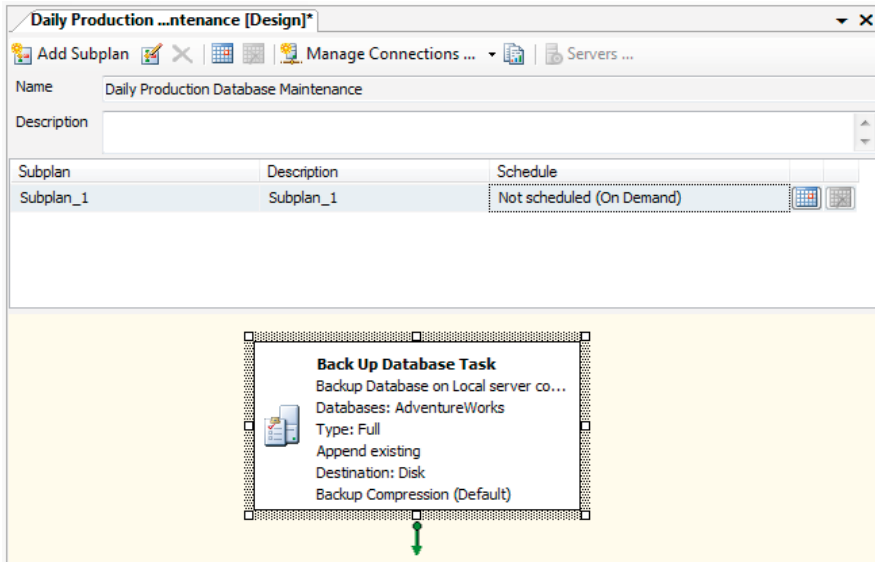

Figure 18.2: The Backup Database Task has been added to Subplan_1, and configured.

The next step is to create the schedule for this subplan by clicking on the calendar icon for **Subplan_1**. The **Job Schedule Properties** screen appears, as shown in Figure 18.3.

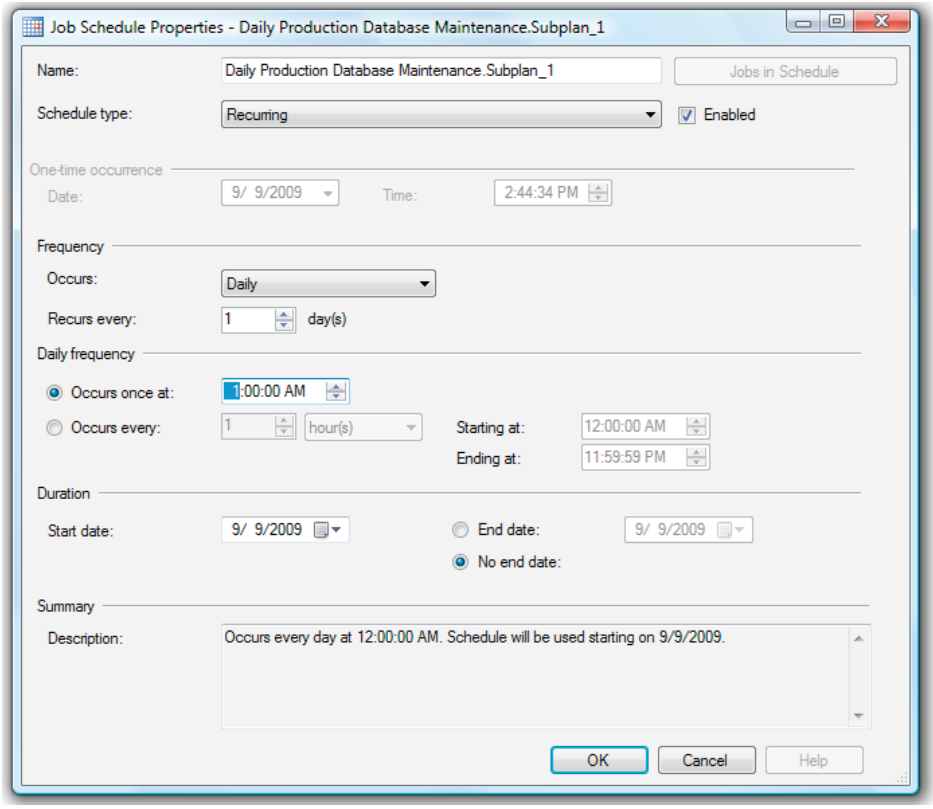

Figure 18.3: This is the Job Schedule Properties screen we have seen many times before.

In Figure 18.3, I have scheduled **Subplan_1**, containing the full backup task, to occur once a day at 1 a.m. Once the schedule has been set, click **OK** to continue, and you will be returned to the Designer, where you will see that the schedule has now been set for **Subplan_1**, as shown in Figure 18.4.

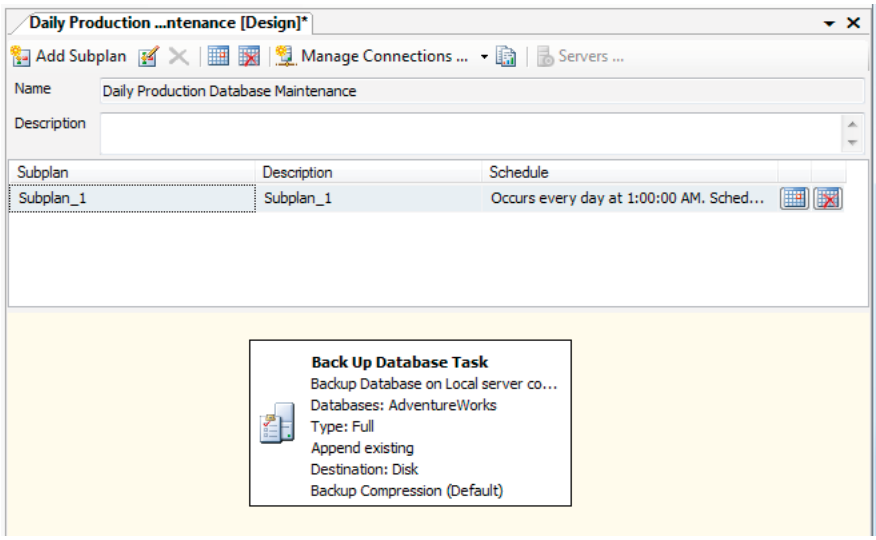

Figure 18.4: The schedule for Subplan_1 has been set.

The next task is to create a second subplan, then add a second instance of the `Back Up Database` task to the new subplan, configure the task to perform transaction log backups, and then to schedule the subplan appropriately. To add a new subplan, click on the **Add Subplan** icon in the Designer menu bar to bring up the **Subplan Properties** screen shown in Figure 18.5.

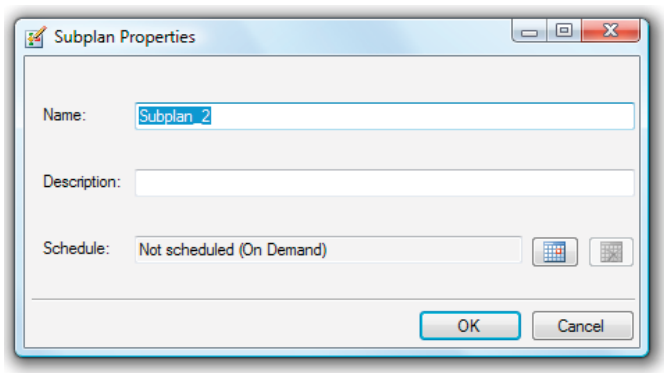

Figure 18.5: The Subplan Properties screen can use all default values, if you wish.

Here, you can enter your own custom name for the subplan, a useful description, and even add the schedule. However, let's keep things simple and simply accept the default values and click **OK**. The Maintenance Plan Designer screen should now list the newly-created subplan, with its default name of **Subplan_2**, as shown in Figure 18.6.

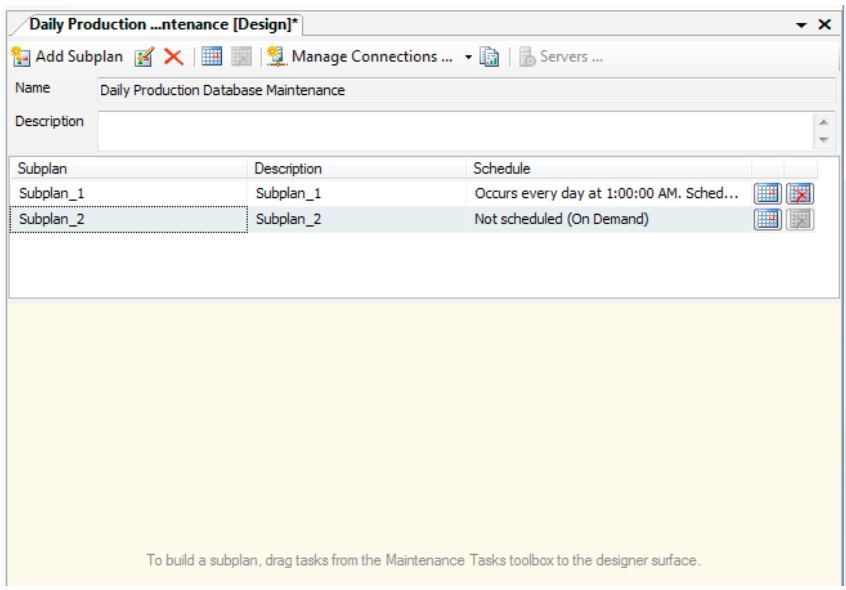

Figure 18.6: Two subplans are now part of this Maintenance Plan.

Click on **Subplan_2** to bring that subplan's design surface into focus, drag and drop another `Back Up Database` task onto it, and then configure it to perform transaction log backups. When done, the screen should look as shown in Figure 18.7.

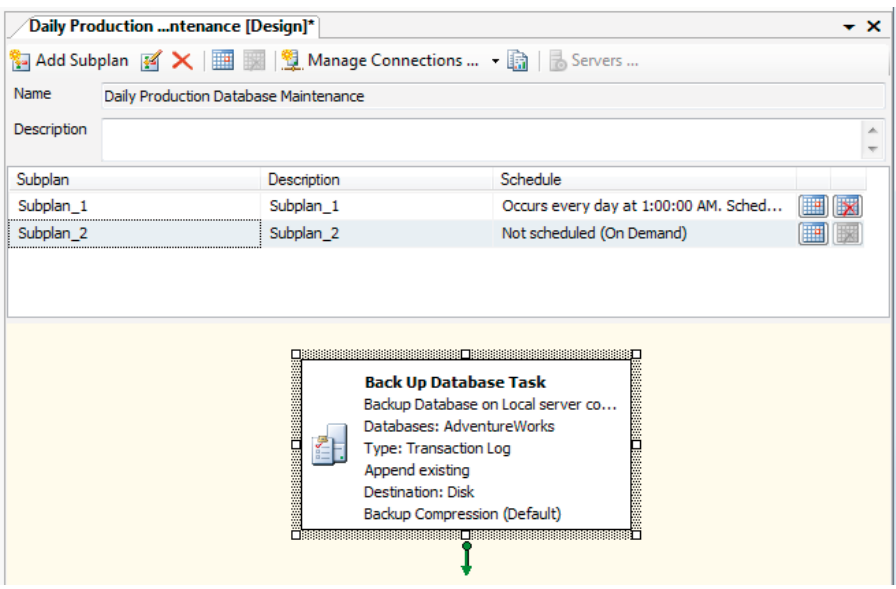

Figure 18.7: The Backup Database Task has been added to Subplan_2 and configured.

Once the `Back Up Database` task has been added to the design surface of **Subplan_2** and configured, you can schedule it as required. In this case, I schedule the subplan to run hourly, every day. The final designer screen looks as shown in Figure 18.8.

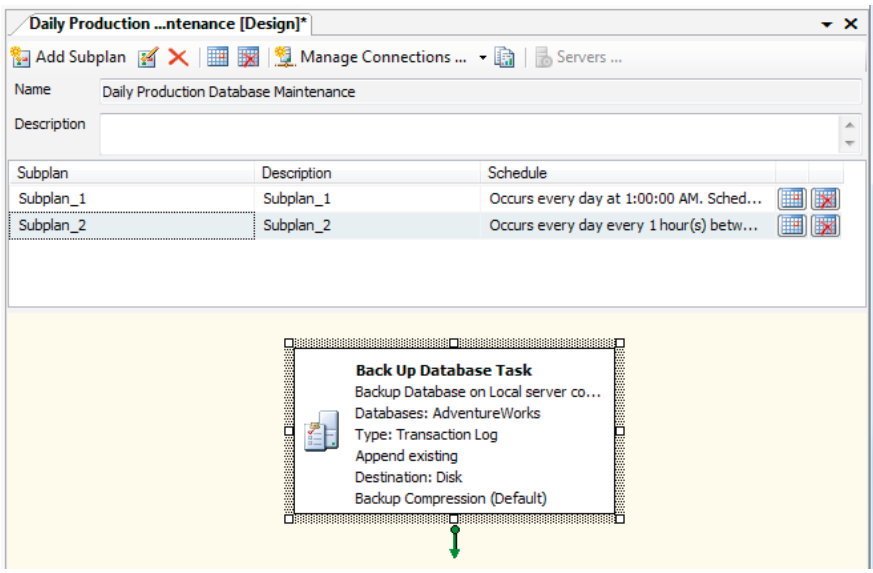

Figure 18.8: This Maintenance Plan, although oversimplified, is ready to run.

As you can see, adding subplans to a Maintenance Plan, and setting the schedule for each subplan, is not a difficult task. However, as I mentioned in the introduction to the chapter, my advice is to keep the number of subplans to a minimum. Instead, place as many tasks as you can in a single subplan, and use precedence links to carefully control their execution.

How to Use Precedence

Precedence is used within a subplan of Maintenance Plan in order to control what happens next in the subplan, based on the outcome of a preceding task. In other words, it's the equivalent of adding conditional logic inside your Maintenance Plan to the effect that "if task A succeeds, execute task B. However, if task A fails, execute task C in place of task B." This powerful "green arrow" feature allows you much more control over your Maintenance Plans than is offered by the Maintenance Plan Wizard.

Task Parallelism in the Designer

Designer also supports a feature called task parallelism (discussed later), which allows you to run two or more tasks in parallel. Based on my experience, I don't recommend this option as it adds complexity, and could potentially cause performance problems if you accidently run two resource intensive maintenance tasks at the same time.

The easiest way to understand how precedence works is to see it in action in a simple, but realistic example. Let's say that we have a simple Maintenance Plan that consists of the following three tasks:

- Back Up Database
- Maintenance Cleanup
- Notify Operator.

The `Backup Database` task should occur first, and will perform a **full** backup of the specified database (`AdventureWorks` in this example). This is the **precedent** task and what happens subsequent to the execution of this task will depend on its outcome. If the `Backup Database` task succeeds, the `Maintenance Cleanup` task should immediately execute, our goal being to preserve only the two most recent backup files on the local SQL Server, and delete any older backup (`BAK`) files. If the `Backup Database` task should fail, we do **not** want to execute the `Maintenance Cleanup` task because it may well be that we need the older backup files to be easily accessible, if the reason the backup failed was because the database had become corrupted and a good backup of it could not be made. It's always wise not to remove old backups from the local server until you're sure that more recent backups are "sound."

In addition, should the full backup fail, we want an e-mail to be sent to an operator so that a DBA can quickly check out what the problem is, and fix it. In other words, if the `Backup Database` task fails, we want the next action to be the execution of the `Notify Operator` task.

As discussed earlier, if you want to establish precedence between a given set of tasks, then each of the tasks must be part of the same subplan and design surface. To start this example, let's drag and drop these three Maintenance Plan tasks onto the default subplan, as shown in Figure 18.9.

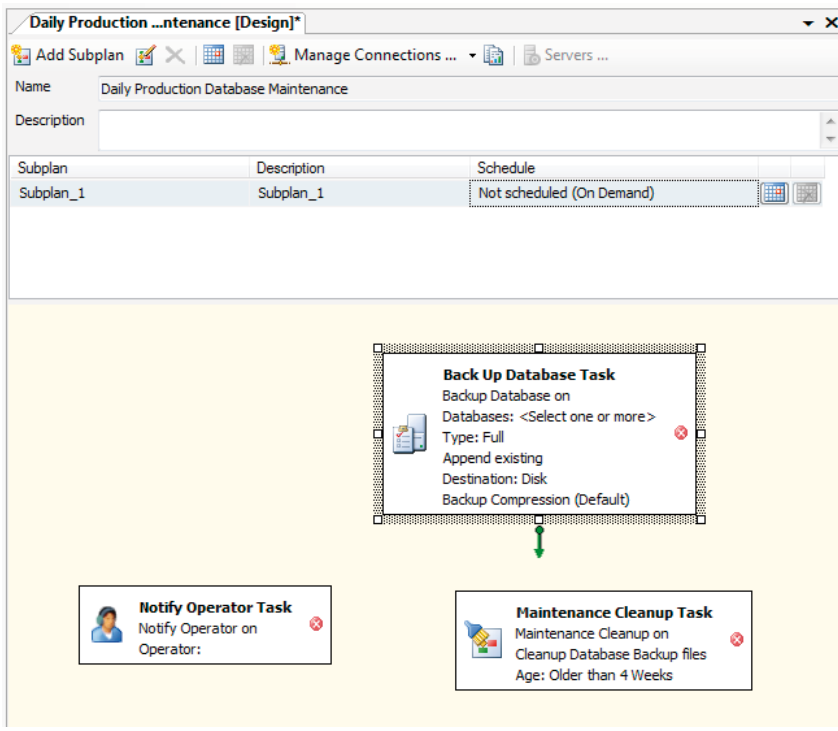

Figure 18.9: Three Maintenance Plan Tasks have been dropped onto a design surface.

Before establishing the precedence of these tasks, the first step is to configure each one appropriately, depending on your needs. I won't cover all the options again for each individual task, so just configure each one as appropriate to our example, so that you have three configured tasks sitting on the surface, not as yet related to each other in any way.

Now it's time to establish a conditional relationship between them in order to achieve the stated goals of our example. Let's tackle it one step at a time. The `Back Up Database` task needs to run first followed by one of the two dependent tasks. Assuming that the `Back Up Database` task succeeds, then we want the `Maintenance Cleanup` task to run. In other words, we need to establish the precedence that the `Back Up Database` task runs first, and that the Maintenance Cleanup task runs second, assuming that the `Backup Database` task was successful. In order to do this, click on the `Backup Database` task so that it has focus and then drag and drop the green arrow from that task directly onto the `Maintenance Cleanup` task. The screen should now look as shown in Figure 18.10.

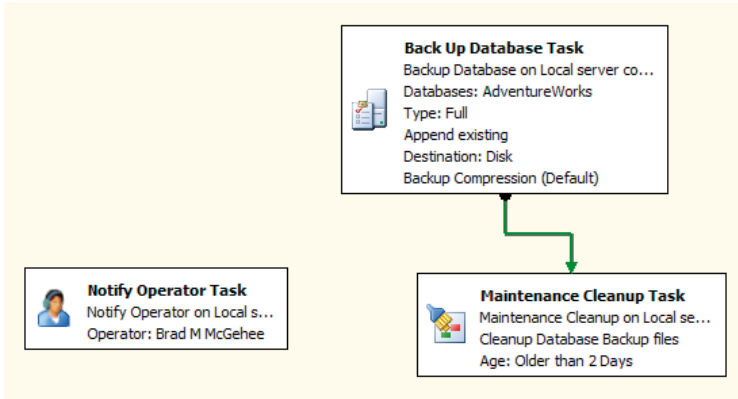

Figure 18.10: Notice that the green arrow starts at the Backup Database Task and points at the Maintenance Cleanup task.

The arrow always must originate at the precedent task, `Back Up Database`, which will execute first, and terminate at the dependent task, `Maintenance Cleanup`, which will execute second, depending on any imposed conditions. The conditions are imposed inside the precedence arrow itself and, in fact, the green color of this arrow in Figure 18.10 indicates that the arrow is, by default, imposing an "on success" condition on the execution of the `Maintenance Cleanup` task. In other words, the condition can be expressed as follows: "On success of the backup task, execute the cleanup task."

How do we verify this? If you right-click on the green line and then select "Edit," or double-click on the green line, the **Precedence Constraint Editor** screen appears, as shown in Figure 18.11.

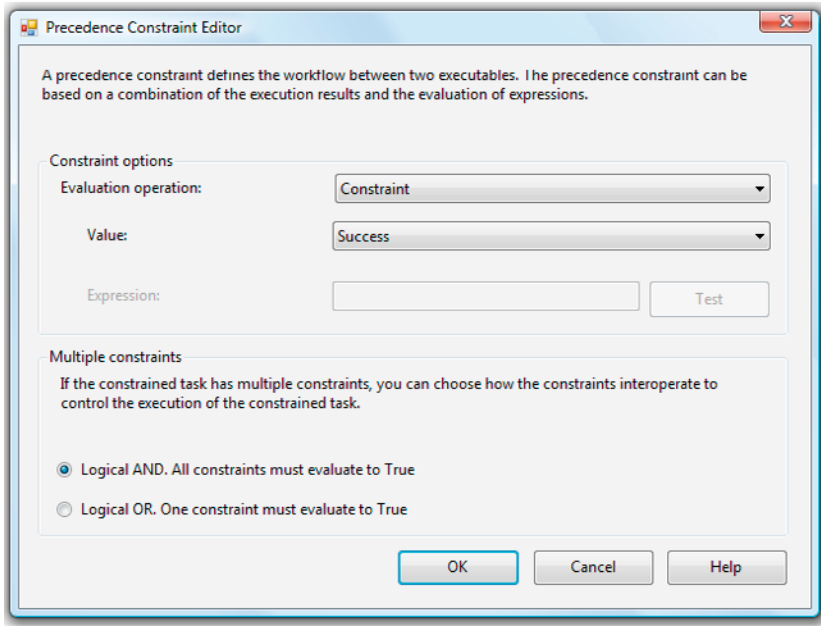

Figure 18.11: The Precedence Constraint Editor is used to help you establish the types of precedence that are available.

There are quite a few options on this screen, but it turns out that we can ignore most of them. The **Precedence Constraint Editor** screen looks more complicated than it really is, because it includes options that aren't really applicable to creating Maintenance Plans. Like much of the code used by the Maintenance Plan Designer, it is reused in different parts of SSMS, and so certain options are presented out of context. For example, the **Expression** option, available as an **Evaluation operation**, is really designed to create SSIS packages, not Maintenance Plans.

The only option with which we really need to be concerned is **Value**. By default, **Success** is selected in the drop-down list and this is the origin of the green color of the arrow in Figure 18.10. Only if the `Back Up Database` task executes successfully will the `Maintenance Cleanup` task execute.

There are two other options available in the **Values** drop-down list, namely **Failure** and **Completion**, as shown in Figure 18.12.

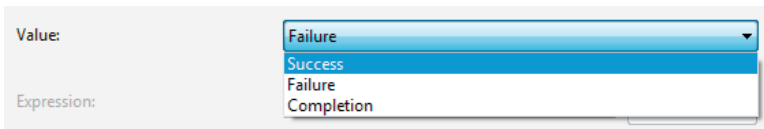

Figure 18.12: We need only focus on three Values.

To impose an "on failure" condition, simply select **Failure** from the list and the resulting precedent arrow will be red. To impose an "on completion" condition, select **Completion**, and the resulting arrow will be blue. The **Completion** condition stipulates that the dependent task should run as soon as the precedent task completes execution, regardless of whether it succeeded or failed. In most cases, you will find the success and failure conditions more useful.

So far, we have only implemented half of our required logic. At the moment, if the backup task succeeds, the cleanup task will be executed. However, if the backup task fails, we want this event to immediately send a notification e-mail to the DBA (operator), via execution of the `Notify Operator` task. To do this, first click on the `Back Up Database` task so that it is selected (in focus), and a second green arrow appears on the `Back Up Database` task box, as shown in Figure 18.13.

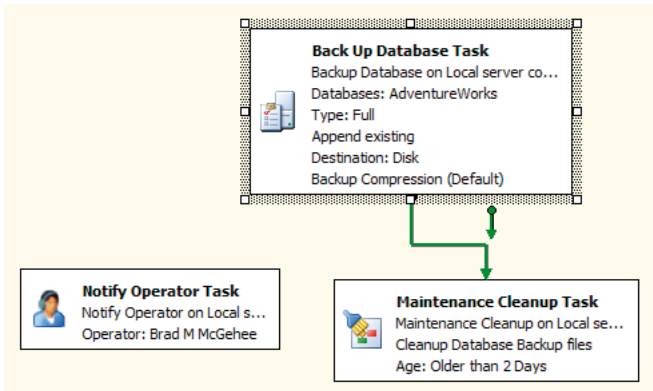

Figure 18.13: A task can have multiple precedence arrows.

Once the second green arrow is displayed, drag and drop it on to the `Notify Operator` task box on the design surface, as shown in Figure 18.14.

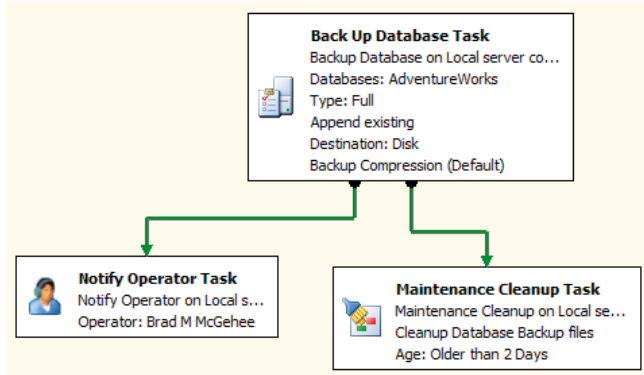

Figure 18.14: Now the Back Up Database Task has two precedence arrows.

As it stands now, in Figure 18.14, if the backup task succeeds then both the cleanup task and the notification task will be executed at the same time.

Earlier, I referred to task parallelism, and recommended that you not use it, due to the complexity, and potential performance issues it brings; and it is not the desired behavior in this example. We need to change the existing green (success) between the Backup Database task and the Notify Operator task to a red (failure) arrow so that precedence replaces task parallelism.

To do this, double-click on the arrow leading to the `Notify Operator` task, change the **Value** to **Failure** and click **OK**. This will impose the required "on failure" condition and change the green to a red arrow, as shown in Figure 18.15.

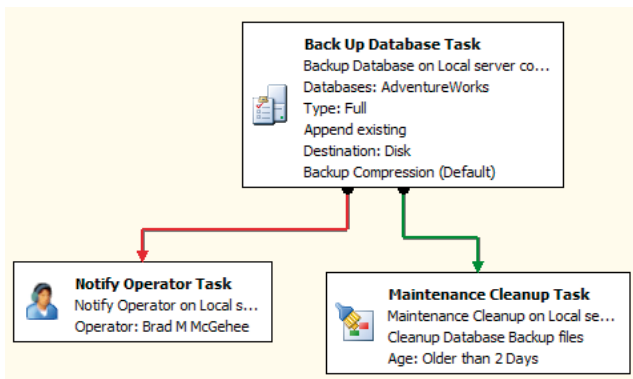

Figure 18.15: The red arrow represents failure.

We are now done. If the `Back Up Database` task executes successfully, the `Maintenance Cleanup` task will be executed. If the `Back Up Database` task fails, then the `Maintenance Cleanup` task will not execute. Instead, the `Notify Operator` task will execute, sending an e-mail message to the designated operator.

Summary

Subplans are a necessary and useful feature, although I recommend you use them as sparingly as possible in order to avoid overcomplicating your Maintenance Plans. Precedence is a very powerful tool, assuming that it is correctly used. It allows you to add logic to your Maintenance Plans, and include error-trapping, of sorts. In the next (and final) chapter, where we create an entire Maintenance Plan from scratch, we investigate additional examples of how precedence can be used.

Chapter 19: Create and Modify Maintenance Plans Using the Designer

It has been quite a long journey through the Designer to get to the point where we are ready to design, create, test, and schedule a full Maintenance Plan.

In this chapter, we'll walk through the full process of creating from scratch, and deploying, a Maintenance Plan that could be used to perform the following essential database maintenance tasks:

- back up database data and log files
- run regular database integrity checks
- perform index rebuilds
- delete old data from the `msdb` database
- remove old backup and report files that are no longer required to be stored on the local server
- notify the designated operator should one of the tasks fail.

While the example does perform many of the essential maintenance tasks, it's not intended as a "template" for how to create Maintenance Plans, nor does it cover all the necessary maintenance tasks. The exact nature of a Maintenance Plan will always depend on the exact nature of your business and administration needs. You must establish exactly what your Maintenance Plans need to achieve, and then implement them appropriately, using the available options that best meet your needs.

Establishing Your Maintenance Goals

Before you can create a Maintenance Plan for your SQL Server instances, you have to first establish what you want to accomplish. Based on established goals, the DBA must create Maintenance Plans that include the appropriate tasks, appropriately configured and scheduled, to meet these goals.

To some extent, these goals will be established at an organizational level. For example, tolerance towards potential data loss should be established at a business level and on a system-by-system basis. This, in turn, will dictate the DBA's maintenance policy in regard to the nature and frequency of database backups in his or her Maintenance Plans. Elsewhere, the nature of the plans will be guided by the DBA's knowledge of a given system, of its databases, the data they contain, how that data is queried, how indexes are used, and so on. However, all plans must start somewhere and, if you don't have all the information available to make an informed decision, you'll need to create the plans using the data and knowledge you do have available, and then monitor them. As your knowledge of a system grows, so your plans can evolve and become more efficient. What's most crucial is that these essential maintenance tasks **do** get performed on a regular basis.

Following is an example list of the objectives of a Maintenance Plan designed to maintain the `AdventureWorks` database on a SQL Server instance.

- Once a day, use the `Back Up Database` task to perform a full backup on the `AdventureWorks` database.
- Every hour, use the `Back Up Database` task to perform a transaction log backup on the `AdventureWorks` database.
- Every Sunday, during a scheduled maintenance window, perform the following tasks:
    - Run the `Check Database Integrity` task
    - Run the `Rebuild Index` task
    - Run the `History Cleanup` task, deleting files older than 1 week
    - Run the `Maintenance Cleanup` task as follows:
        - Delete `BAK` files older than 2 days
        - Delete `TRN` files older than 2 days
        - Delete report text files older than one week.
- If any of the previously listed tasks should fail, execute the `Notify Operator` task to immediately send an e-mail that tells the operator what task failed.
- If any of the tasks fail, stop the execution of the Maintenance Plan so that any subsequent tasks aren't executed. This means that the operator has the opportunity to fix a problem before remaining tasks are executed.

Having established the identity and nature of the database maintenance tasks that need to be carried out, we need to translate them into an actual Maintenance Plan.

Creating Maintenance Plans: the Big Picture

While there are many ways in which to translate database maintenance goals into an actual Maintenance Plan, I like to follow the ten steps below.

1. Create the new Maintenance Plan.
2. Create any necessary subplans.
3. Add the Maintenance Plan tasks to each subplan, as appropriate.
4. Configure each Maintenance Plan task.
5. Establish the necessary precedence links between tasks in a given subplan.
6. Define reporting and logging requirements.
7. Save the Maintenance Plan.
8. Test the Maintenance Plan.
9. Set the subplan schedules.
10. Run in production, and follow up.

While you don't have to follow these steps in this exact order (the Maintenance Plan Designer is very flexible) I think you will find this particular order will help you manage the process most effectively, when you are first starting out using the tool.

Create the New Maintenance Plan

Before we can do anything, we must first create a new Maintenance Plan using the Maintenance Plan Designer. Right-click on **Maintenance Plans**, within the **Management** folder of SSMS Object Explorer, and select **New Maintenance Plan**. Enter a descriptive name in the **New Maintenance Plan** dialog box, and click **OK**. The new (empty) Maintenance Plan is created, as shown in Figure 19.1, and we are ready to begin.

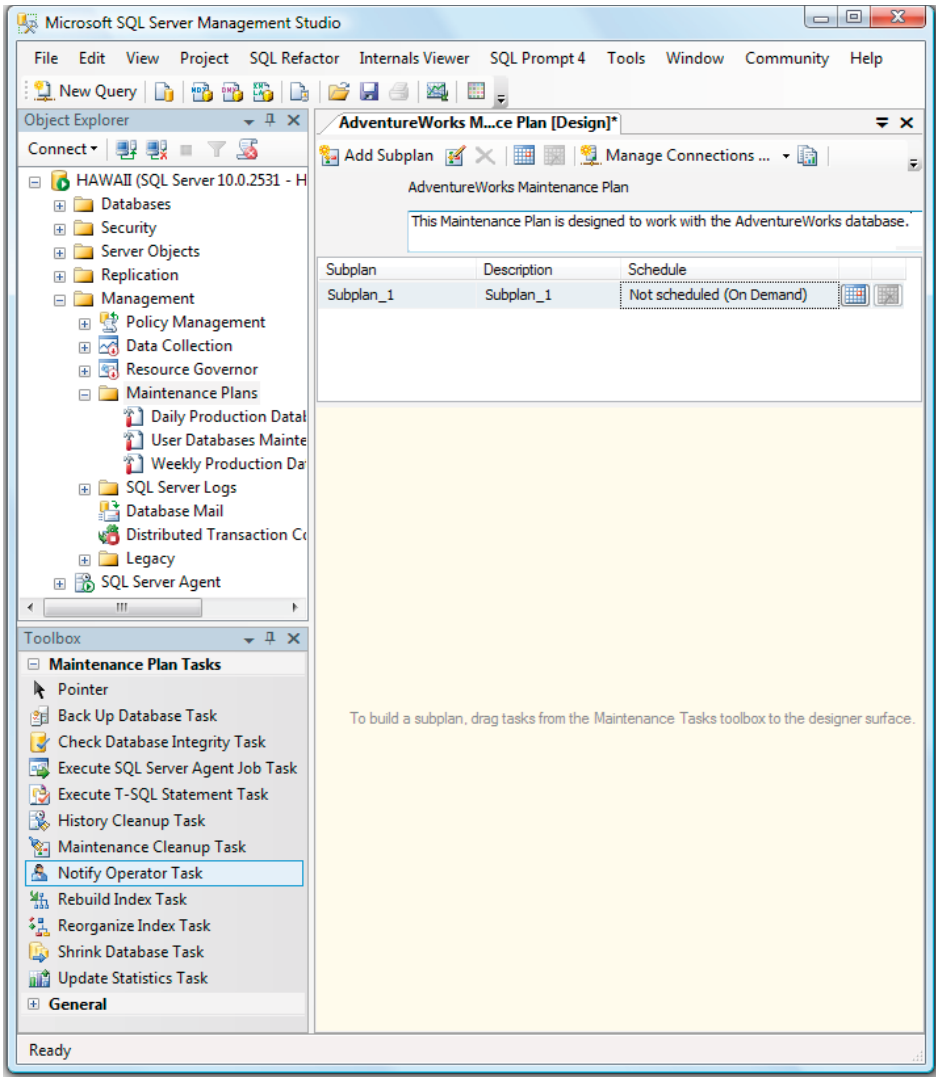

Figure 19.1: Once you have created the Maintenance Plan, you are now ready to build it.

In this example, the new plan is called **AdventureWorks Maintenance Plan**. Optionally, we can add a description of this plan in the dialog box immediately below its name. Once this initial step is completed, we are ready to create any subplans appropriate to the goals of our Maintenance Plan.

Create the Subplans

As you may recall, the purpose of a subplan is to separate Maintenance Plan tasks into different schedules, as required. So, how many subplans do we need to meet our database maintenance goals? Did you guess right? We need three subplans, because our maintenance goals include three different scheduling requirements.

- **Once a day** – for full database backups.
- **Once an hour** – for transaction log backups.
- **Once a week** – for all other database maintenance tasks.

Since there is already one default subplan, we need to create two more, so add these to the Designer using the **Add Subplan** button in the menu bar. While your subplans will automatically be assigned non-descriptive names, it is always wise to make each plan as self-documenting as possible, as it will make it much easier for another DBA to review it and understand how it works. Having created the extra subplans, and given all three subplans descriptive names, the screen should look as shown in Figure 19.2.

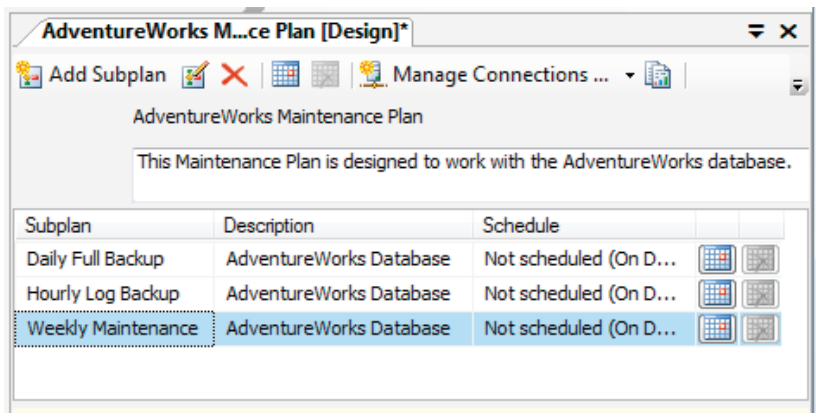

Figure 19.2: Three subplans have been created

When you add subplans, they appear in the order that you create them, and you can't change that order. Fortunately, this is not a problem, because each of these subplans operates on a different schedule, so their order on the screen is irrelevant.

The presence of those calendar buttons next to each subplan may tempt you into assigning the schedules now. However, resist the temptation. If you schedule the subplans, and then subsequently save the unfinished plan (in order to preserve your progress), the underlying SSIS packages and Agent jobs will be created, and SQL Server may attempt to start running the scheduled jobs! Never schedule the subplans until the plan is complete and tested, and

you're ready to start using it.

Add the Maintenance Plan Tasks

At this point, we have three subplans, each with its own design surface. Our next task is to drag and drop the appropriate Maintenance Plan tasks to the design surface of the appropriate subplan. Let's consider each subplan in turn.

Daily Full Backup Subplan

According to the goals of our overall plan, this subplan should contain:

- One `Back Up Database` task, to perform the full backup of AdventureWorks.
- One `Notify Operator` task, to notify the operator, should the full backup fail.

Highlight the **Daily Full Backup** subplan and then drag each of these two tasks from the Toolbox and drop them on the design surface, as shown in Figure 19.3.

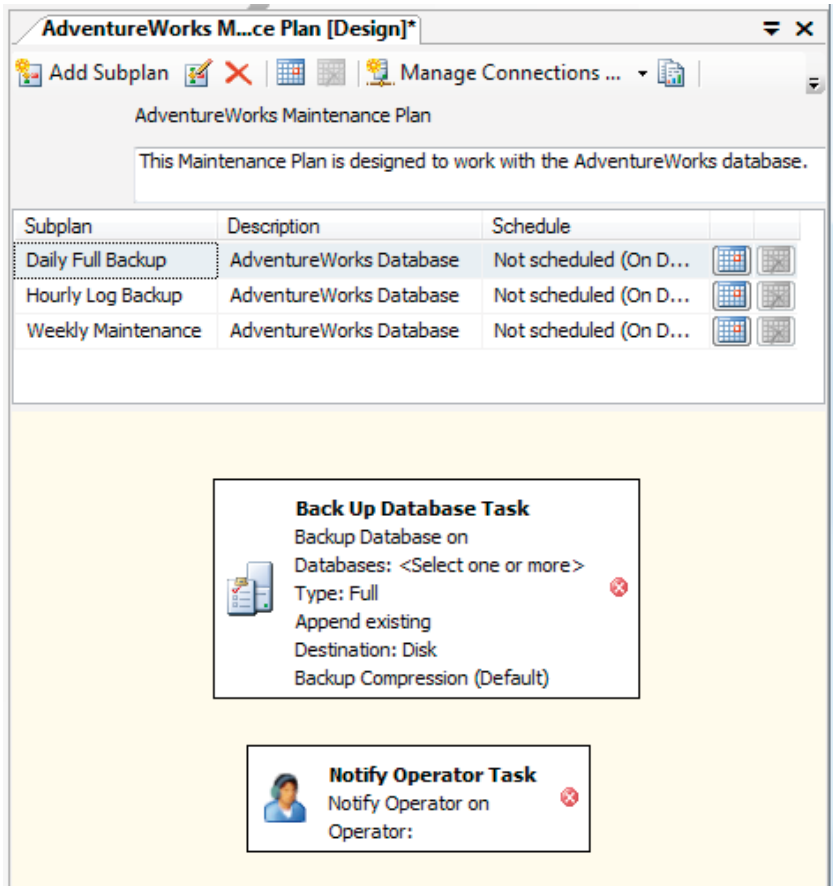

Figure 19.3: The Daily Full Backup subplan now has the necessary Maintenance Plan Tasks.

Note, of course, that although we've now added the appropriate tasks, they are not yet configured. We will do this later, after adding the required tasks to the other two subplans.

Hourly Log Backup Subplan

According to our goals, this subplan should contain:

- One `Back Up Database` task – to perform hourly backups of the transaction log for the AdventureWorks database.
- One `Notify Operator` task – to notify the DBA, should the transaction log backup fail.

The resulting design surface should look as shown in Figure 19.4.

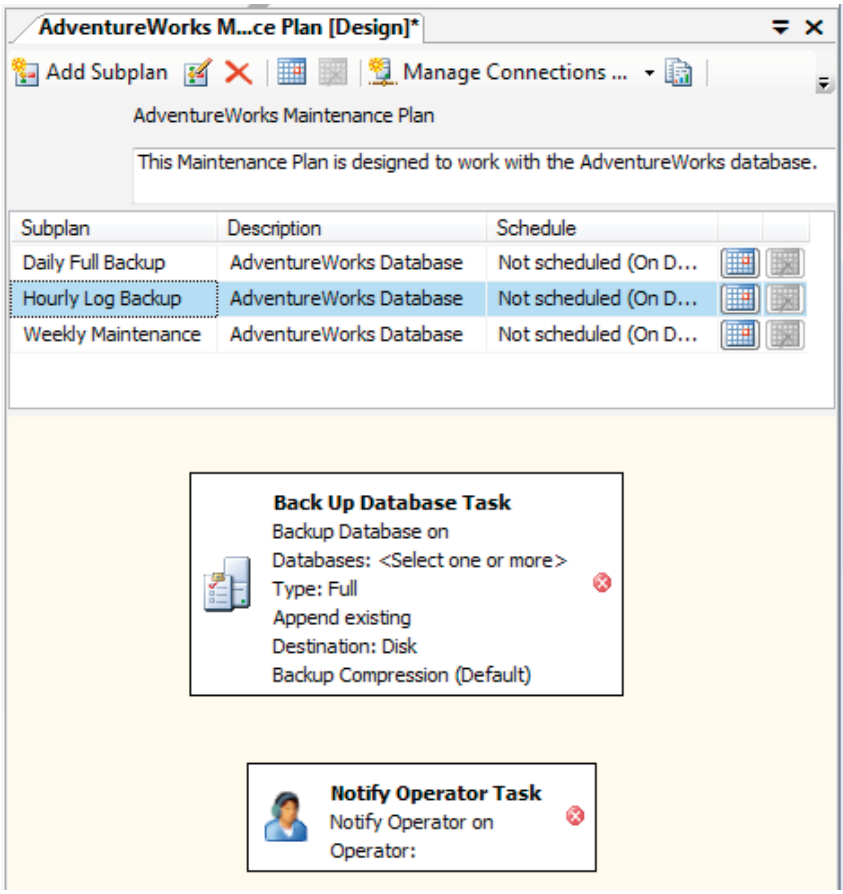

Figure 19.4: Notice that the Hourly Log Backup subplan now looks like the Daily Full Backup subplan.

In their unconfigured states, the **Daily Full Backup** and the **Hourly Log Backup** subplans look identical. Later, when we configure them, we will specify one to do a full backup and one to do a transaction log backup.

Weekly Maintenance Subplan

Finally, we need to add Maintenance Plan tasks to our most complex subplan, the **Weekly Maintenance** subplan. Based on our database maintenance plan goals, we will need to add the Maintenance Plan Tasks below to the Weekly Maintenance subplan.

- One `Check Database Integrity` task – to check the integrity of AdventureWorks.
- One `Rebuild Index` task – to rebuild all of the indexes in AdventureWorks.
- One `History Cleanup` task – to remove old backup, job, and maintenance plan history data from `msdb`.
- Three `Maintenance Cleanup` tasks – each instance of this task can only delete one file type, so we need a total of three instances in order to delete older `BAK`, `TRN`, and report TXT files.
- Seven `Notify Operator` tasks – one instance for each of the previous six Maintenance Plan tasks, to let the operator know if the task failed, plus an extra instance to let the operator know that the entire subplan succeeded.

The resulting design surface would look similar to that shown in Figure 19.5.

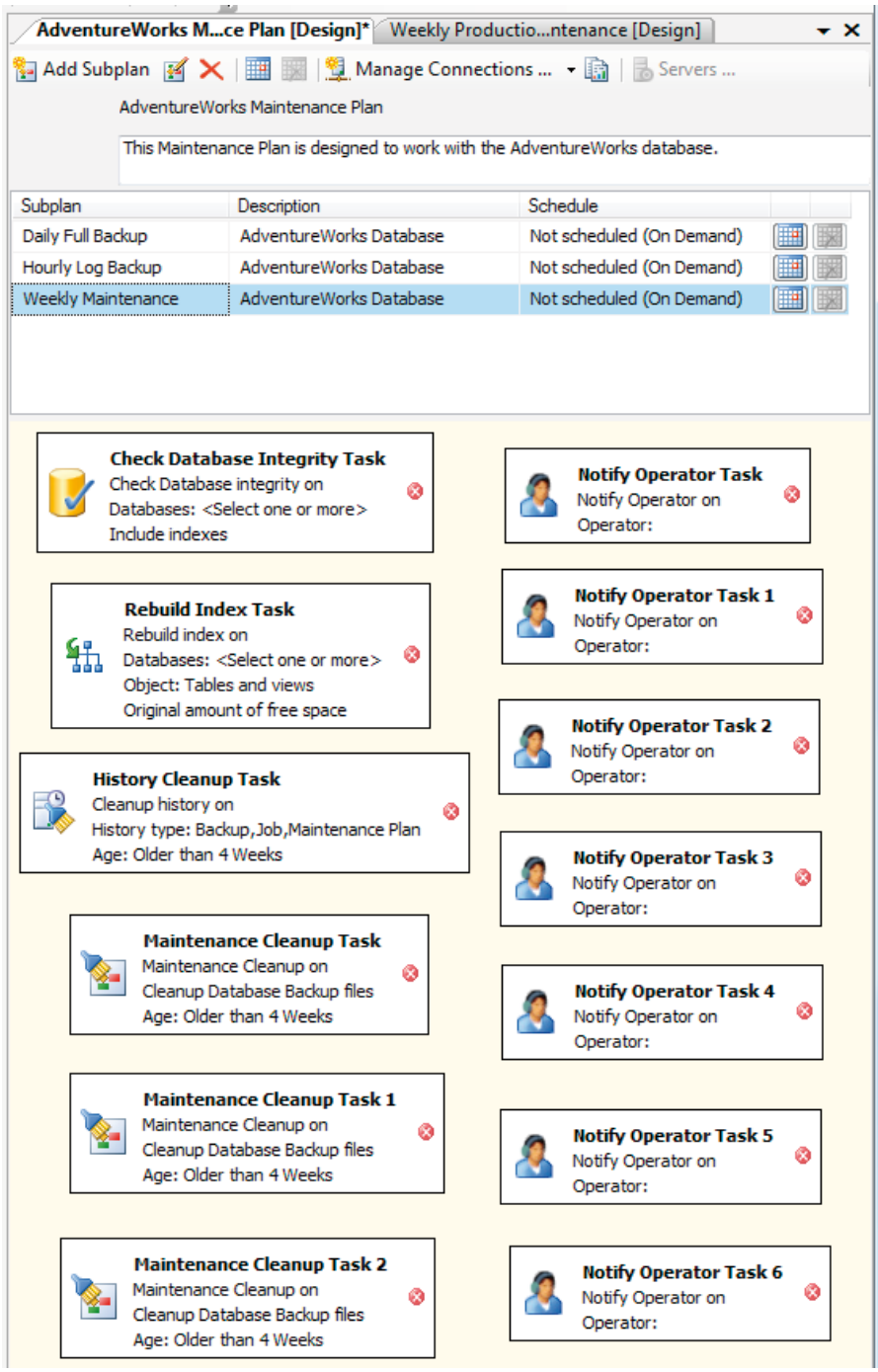

Figure 19.5: The Weekly Maintenance subplan is the most complex of the three.

The **Weekly Maintenance** subplan is beginning to look crowded, but it actually looks more complex than it is, partly due to the fact that I included so many `Notify Operator` tasks. These tasks are entirely optional. You might want to use fewer of them, perhaps only receiving notification of the success or failure of those tasks you deem most critical, but I like to be notified if any one of the steps within the subplan fails. I also find it reassuring to receive a mail confirming that the whole plan ran successfully, hence the seventh instance of the task.

Configure the Maintenance Plan Tasks

Having assigned all the required Maintenance Plan tasks to their relevant subplans, the next step is to configure each task appropriately. Having covered all the options for each task previously, either in Chapter 17 (or in each task's namesake chapter) when I was discussing the Wizard, I'm not going to repeat myself here.

As you configure each of the task instances on each of the subplans, I recommend that, where needed, you assign each of the Maintenance Plan tasks a more descriptive name, so that it will be easier for you and others to fully understand the intent of the plan. For example, Figure 19.6 shows the **Weekly Maintenance** subplan with each task configured, and many of the tasks renamed.

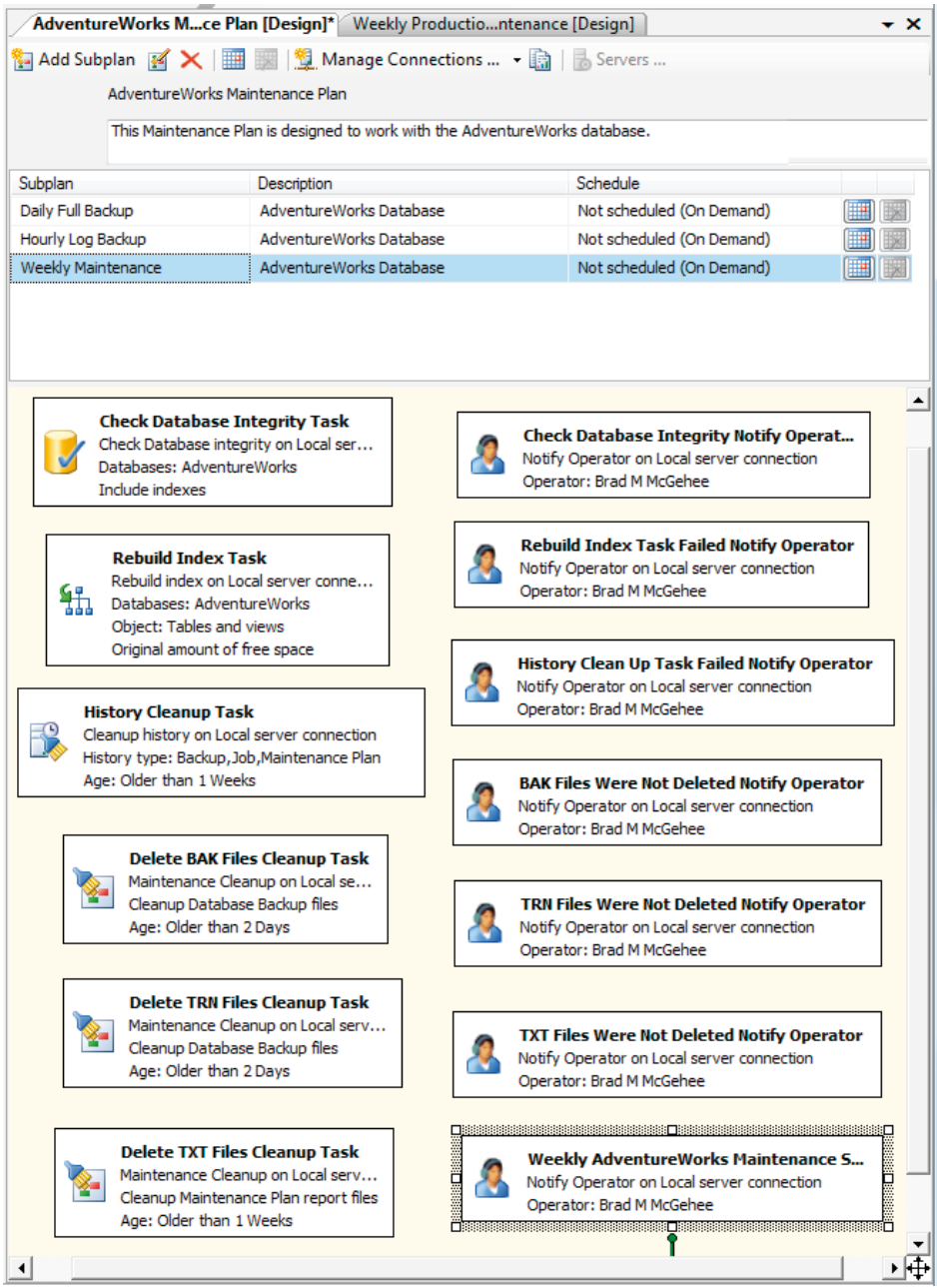

Figure 19.6: Notice the scroll bars and the four-arrow icon on the screen above. As your plan becomes more complex, you may extend it past the boundaries of your screen. For large plans, use the four-arrow icon to move around the plan.

Once all the tasks in all the subplans have been configured, and renamed as appropriate, it is time to set the precedence for each of the tasks within each subplan.

In the configuration screen for each `Notify Operator` task, you should insert a relevant subject line and a short message that will succinctly explain to the DBA why he or she has received that particular e-mail notification.

Note that an alternative scheme would have been to create a generic `Notify Operator` task, and link each of the six core maintenance tasks to this one `Notify Operator` task. However, the resulting e-mail would need to contain a generic subject line and message, and wouldn't be able to tell you which Maintenance Plan task failed.

Set Precedence

In Chapter 18, we walked through a simple example of how to use precedence links, and the conditional logic they contain, to establish the required workflow for a set of three maintenance tasks. Here, our task is more difficult as we are dealing with many more tasks, but the principles are exactly the same.

Let's consider each subplan in turn and establish the precedence relationships that must exist between the tasks in that subplan.

Daily Full Backup Subplan

This simple subplan only has two tasks, `Back Up Database` (performing a full backup) and `Notify Operator`. The `Back Up Database` takes precedence and, if it succeeds, then this subplan has done all the work it needs to do.

> ***Notification on Success***
>
> *We could also add a second* `Notify Operator` *task to this subplan, to notify the operator that the backup task had succeeded. Some DBAs like to receive these notifications, but we've left it out of this example.*

However, if the backup fails, we want the subplan to perform an extra task, and that is to notify the operator of the failure. To create the required precedence for these two tasks, we need to create an arrow between the two tasks and then edit the link (double-click on the arrow) so that it applies an "on failure" condition. The resulting subplan is shown in Figure 19.7. Notice the red arrow.

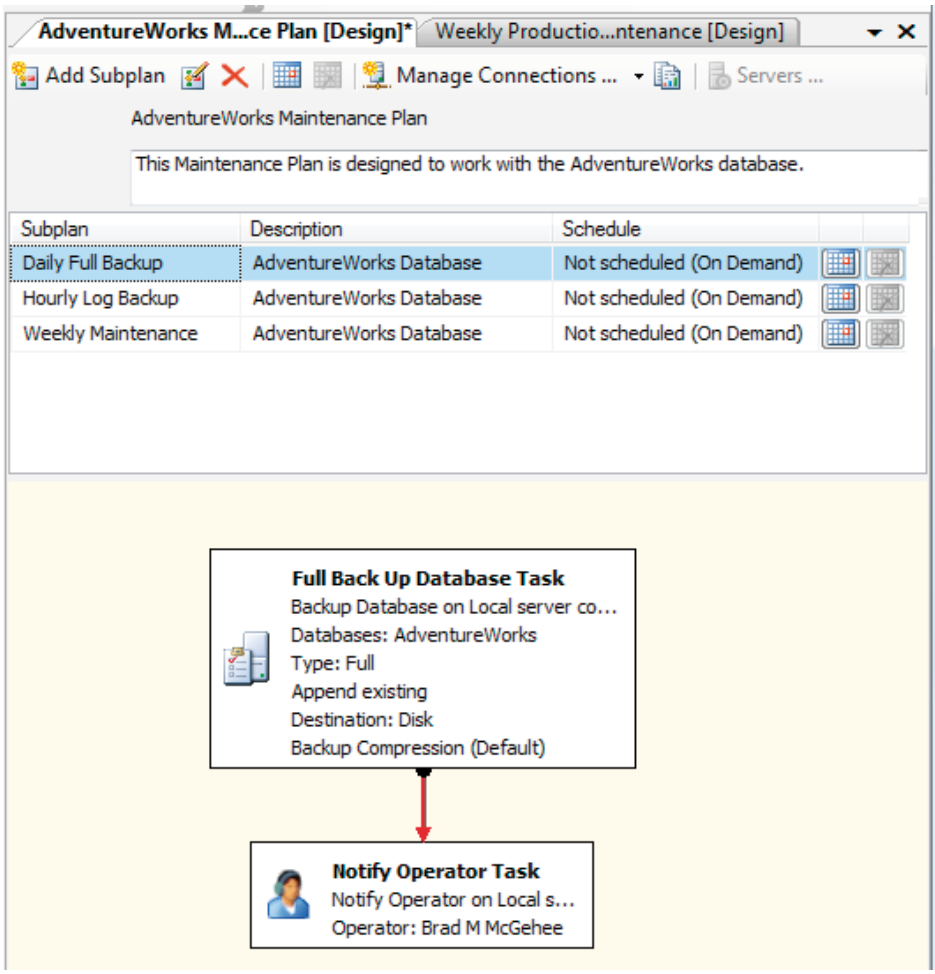

Figure 19.7: Precedence has been set up in the Daily Full Backup subplan

Hourly Log Backup Subplan

The **Hourly Log Backup** subplan is almost identical to the **Daily Full Backup** subplan. If the transaction log back up succeeds, then the subplan's work is done. If it fails, then the `Notify Operator` task executes and notifies the operator of the failure. Figure 19.8 shows the resulting subplan, with the desired precedence established.

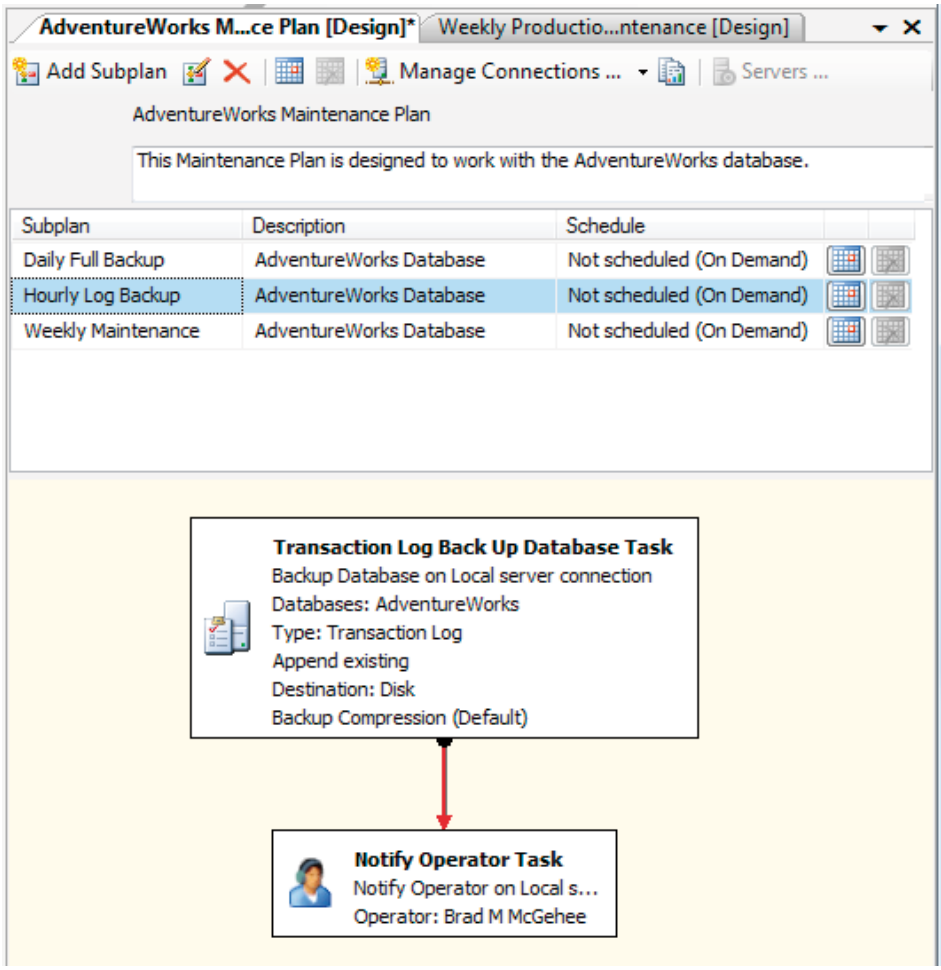

Figure 19.8: The precedence for the Transaction Log Back Up and the Daily Full Back tasks are identical.

Weekly Maintenance Subplan

Due to the many tasks within the Weekly Maintenance subplan, configuring precedence is a little more complicated. Fortunately, it looks worse than it really is. This is because, as long as you keep focused on what you are doing, then the complicated precedence lines that you will soon see won't seem that confusing. In the following example, I will focus only on the key things you need to do. I won't bother repeating myself, as many of the steps are repetitive.

The first step is to link all of the six database core maintenance tasks to establish the order in

which they are to execute. The easiest way to do this is to first order the tasks on the design surface in the proper order, and then draw the green connecting arrows. Setting precedence is always done two tasks at a time, so start with the first two tasks and work your way down. So, for example, we start out by linking the `Check Database Integrity` task and the `Rebuild Index` task. Next, link the `Rebuild Index` task with the `History Cleanup` Task, and so on, until all of the six tasks are linked, as shown in Figure 19.9.

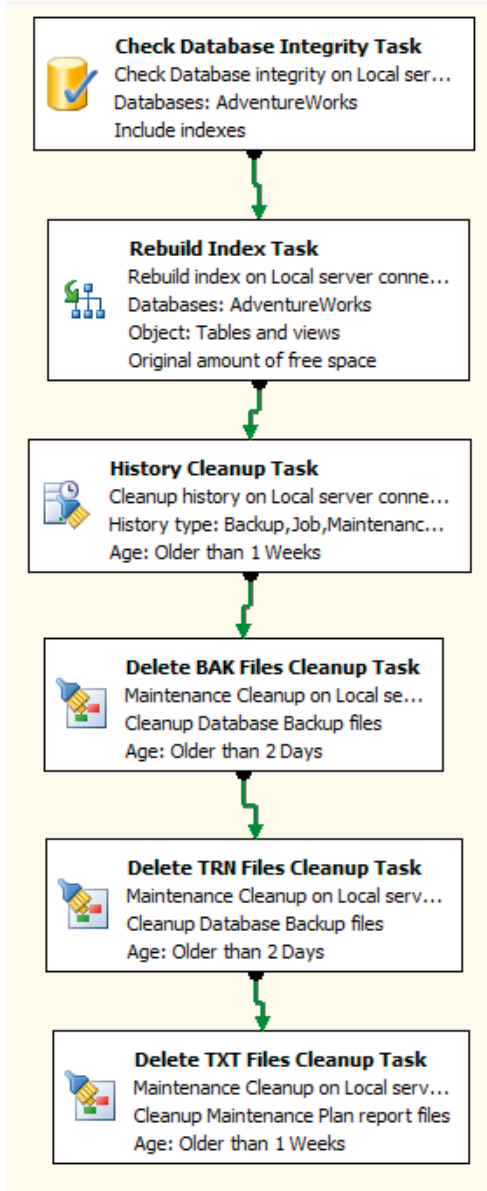

Figure 19.9: The green arrows aren't pretty, but they don't need to be.

When this subplan is scheduled to run, these six tasks will run, one after another, in the order dictated by the direction of the precedence arrows. Of course, this assumes that each of the tasks succeeds. The fact that we are using "on success" conditions (green arrows) to link successive tasks means that a failure of any one of these tasks will prevent tasks further down the chain from executing.

Therefore, if any one of these tasks fails, we want the operator to know about it and act on it. This is where the `Notify Operator` tasks come into play. The next step is to link each of the six core maintenance tasks with one instance of a `Notify Operator` tasks, using a red arrow, indicating an "on failure" condition. Having completed this step, the subplan should look as shown in Figure 19.10.

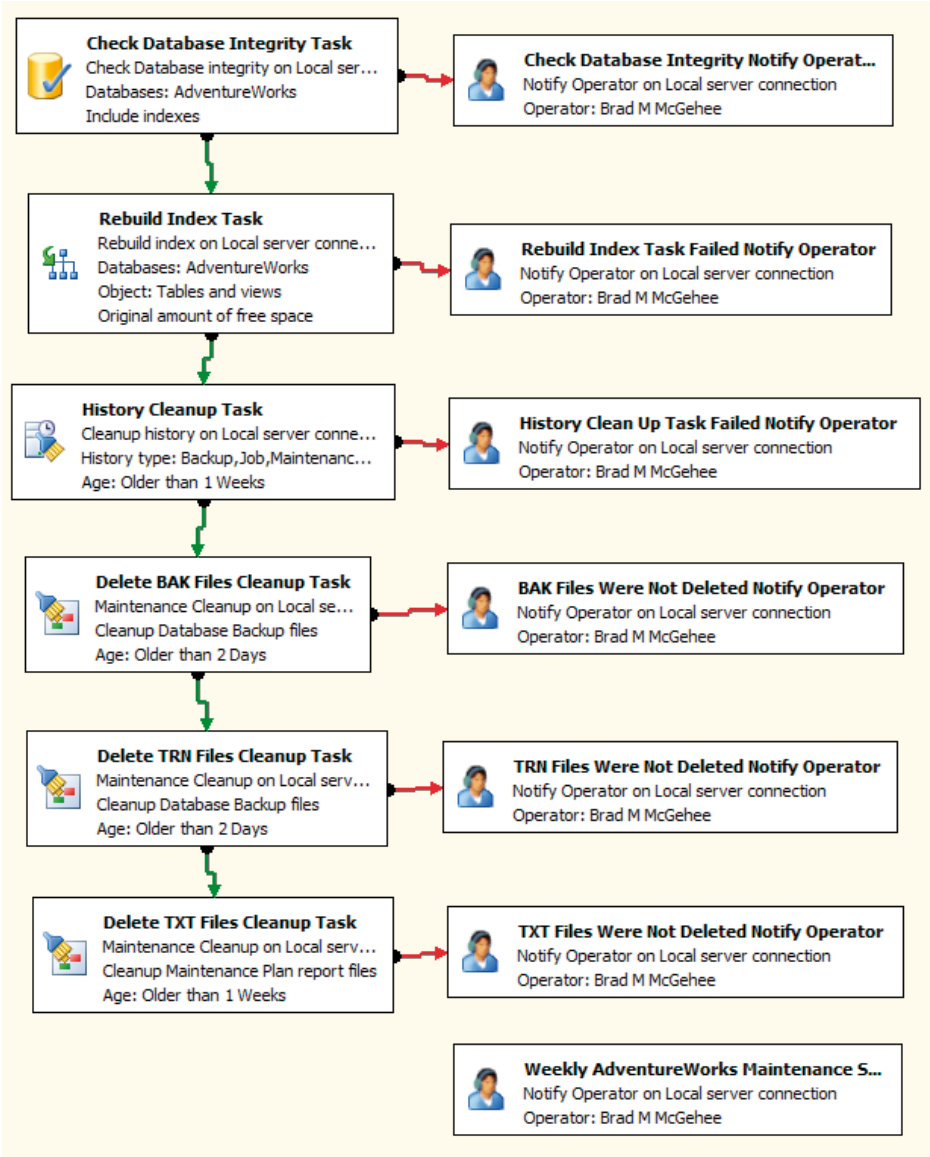

Figure 19.10: The red arrows indicate failure.

Now, if any of the Maintenance Tasks fails, the linked `Notify Operator` task will execute, sending a customized e-mail to the operator, describing which task failed, and explaining that, as a result, the rest of the tasks within this subplan will not execute.

So, if the `Check Database Integrity` task succeeded, but the `Rebuild Index` task failed, the operator would receive an e-mail with a message saying that the `Rebuild Index` task failed, and that the remaining Maintenance Plan tasks would not be executed.

Proceeding in spite of task failure

If you want, you can configure the precedence in such a way as to continue the execution of all the tasks in the subplan, even if one of them fails, instead of cancelling the remaining tasks, as I have done. One way to do this would be to create an "on success" link between the Notify Operator task (that sends the e-mail notifying the operator of a failed task) and the next task you want run next.

Finally, we need to deal with that seventh `Notify Operator` task, which is currently not connected to any other task. Since all our six core maintenance tasks are connected by green arrows, we know that if the last task in the chain, **Delete TXT Files Cleanup**, completes successfully, then all the tasks in the subplan have completed successfully. In this event, our final goal is that the operator receives an e-mail notification to this effect. To implement this final link, drag an arrow from the **Delete TXT Files Cleanup** task to the loan `Notify Operator` task, as shown in Figure 19.11.

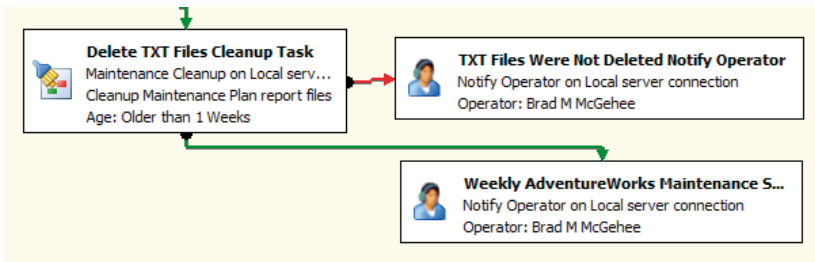

Figure 19.11: The Weekly AdventureWorks Maintenance Successful Operator task will only execute if all six maintenance task succeed.

We have now set up the precedence for all three of our subplans, and we are almost done.

Define Reporting and Logging

The next step in implementing our Maintenance Plan is to click in the Reporting and Logging icon in the designer menu bar and configure our reporting and logging requirements. We discussed the Reporting and Logging options of the Designer in Chapter 16, and I won't go over all the details again here.

However, for our Maintenance Plan, we need to ensure the points below.

- **Generate a text file report** is selected, so a text file report is generated and saved to an appropriate location each time a subplan is executed. In our case, we'll get three reports from this Maintenance Plan, one being written to file every time each subplan runs. These reports will give you the "big picture" of how your plans are executing, and will supplement the e-mails sent via the `Notify Operator` task.

- **The Log extended information checkbox** is checked, so that the data collected as part of the text file reports will be comprehensive, and will make it easier for us to track down any problems with a Maintenance Plan.

The configured Reporting and Logging screen should look similar to the one shown in Figure 19.12.

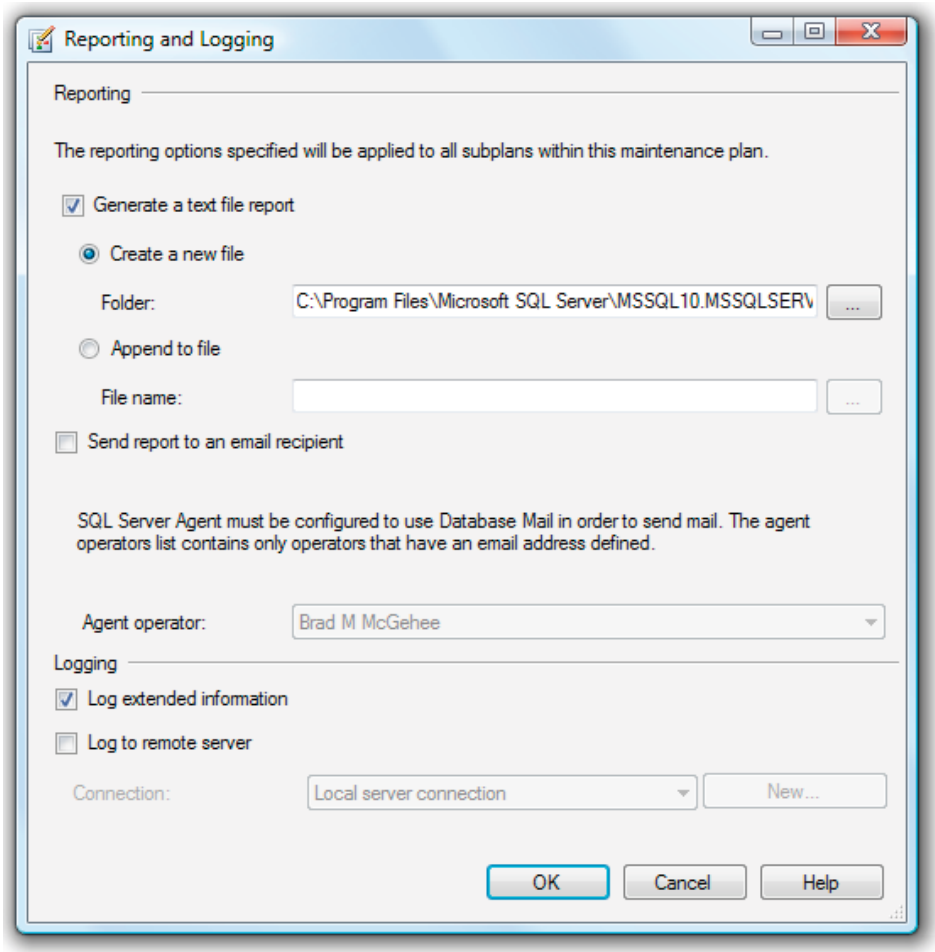

Figure 19.12: Configuring the Reporting and Logging screen can be done any time throughout the Maintenance Plan design process, although I have saved it for last.

For example, in the Maintenance Plan we just created, a report text file will be written to disk every time a subplan executes. The **Daily Full Backup** subplan will be executed once a day (7 reports per week), the Hourly Log Backup subplan will be executed 24 times a day (168 reports per week), and the Weekly Maintenance subplan will be executed once a week (1 report per week). If you need to do any troubleshooting, there will be a lot of text file reports to wade through. Fortunately, each report has a time stamp, and you should be able to narrow down a problem to a specific time period, which will make it easier for you to find any reports for troubleshooting purposes. And as you can see, this is why you need to delete older text file reports as part of your maintenance tasks, as their numbers can add up fast.

Save the Maintenance Plan

Once we are done creating the Maintenance Plan, we can save it by clicking on the **Save Selected Items** icon on the SSMS toolbar. Or, if we try to exit the Maintenance Plan without saving it first, we will be prompted to save the plan. Either method will ensure that the plan is saved.

Test the Maintenance Plan

Having created our Maintenance Plan, it's important to test it before scheduling it and putting it into production. If a Maintenance Plan has only one subplan, it can be started by right-clicking on it and selecting **Execute**. Unfortunately, this does not work if a Maintenance Plan has two or more subplans, which is the case in our example.

Behind the scenes, the Maintenance Plan is implemented as a single SSIS package containing a number of distinct executables, one for each subplan, and each executable is executed using a separate SQL Server Agent job. So, if there is more than one subplan, there is more than one executable and more than one job, and we need to execute each SQL Server Agent job individually in order to test the full Maintenance Plan.

The three SQL Server Agent jobs created for our example `AdventureWorks` Maintenance Plan are shown in Figure 19.13.

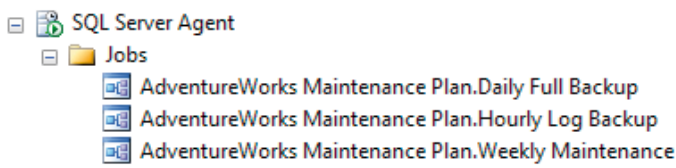

Figure 19.13: Each subplan within a Maintenance Plan has its own associated SQL Server Agent job.

To test each subplan, simply right-click on the appropriate job and select **Start Job at Step**. Remember that we must perform at least one full backup of a database before we can create a transaction log so, for my tests, I ran the **Daily Fully Backup** job first then, once it had completed successfully, the **Hourly Log Backup** job.

Although the **Weekly Maintenance** job could be run first, second or last, I would save it for last because this is normally how the subplans would be scheduled to run once the Maintenance Plan goes into production. As we test each job, a status screen tells us if the job succeeded or failed, as shown in Figure 19.14. This screen only appears when testing the jobs manually and will not be displayed when the jobs run automatically, after they are scheduled.

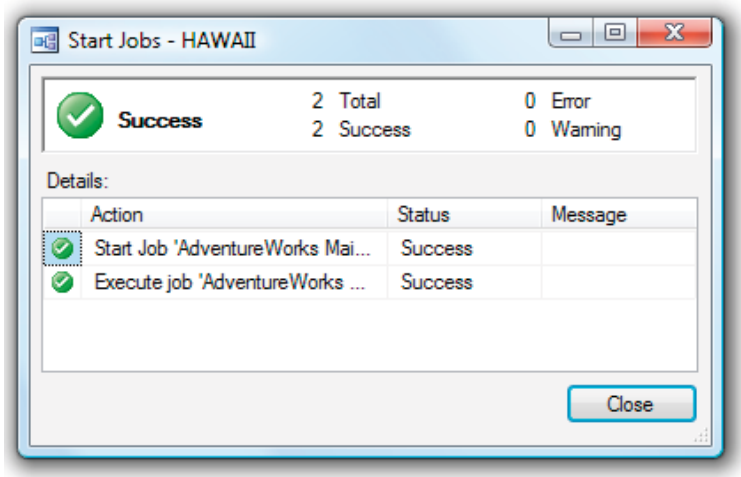

Figure 19.14: Use this status screen to see if the Maintenance Plan subplans succeeded.

If a particular job fails, we'd want to check out the report text file that was created for that subplan. In addition, assuming a given subplan has `Notify Operator` tasks associated with it, the operator should receive an e-mail telling him or her at what specific Maintenance Plan task the subplan failed.

Hopefully, using this information, we'd be able to figure out what the problem was, open the Maintenance Plan using the Maintenance Plan Designer, and make the necessary changes to make the subplan work. Sometimes, multiple test-fix iterations will be required before everything is working correctly.

Once we've verified that all the subplans for our Maintenance Plan run as we expect, we are ready to set their schedules.

Set the Schedules

The final step before putting our Maintenance Plan into production, is to set the schedules for each subplan.

To do this, we'll need to reopen the Maintenance Plan, so right-click on the plan in SSMS Object Explorer and select **Modify**. The Maintenance Plan will open up in the Maintenance Plan Designer.

Next, click on the **Schedule** icon next to each subplan, and set the schedule that is appropriate to meet the goals of the subplan. As previously discussed, the **Daily Full Backup** subplan should be execute once a day, the **Hourly Log Backup** subplan should run once an

hour, and the **Weekly Maintenance** subplan should run once a week, during the weekend maintenance window. See Chapter 4 on scheduling, if you have forgotten how to do this.

Once all the subplans are scheduled, resave your plan, and each subplan begins running immediately, based on those schedules.

Run in Production and Follow Up

Even though I always thoroughly test a Maintenance Plan before putting it into production, I still like to check up on a new Maintenance Plan after it is put into production, to ensure that it is working as expected.

This includes checking to see that the jobs have run, looking at the report text files that have been saved to file, and checking any e-mails that I might have received. I also monitor server resources usage whenever new plans are being run.

When you first schedule your subplans, it is often hard to determine how long a particular subplan will take. If a subplan takes longer than you expected, it may start to conflict with other jobs on your server and so place unnecessary stress on server resources. If this is the case, you may need to reschedule your subplans so that there are no job overlaps.

After a couple of days of checking and monitoring, I leave it up to my plan to let me know if there are any problems.

Modifying an Existing Maintenance Plan

Throughout this book, I have made occasional reference to the fact that you should use the Maintenance Plan Designer to modify Maintenance Plans created using the Maintenance Plan Wizard. This is because the Maintenance Plan Wizard offers no way to modify a Maintenance Plan after it has been created, so the only way to change it *safely* is to use the Maintenance Plan Designer.

I have left the discussion of this topic until now, as you really need to understand how to use the Maintenance Plan Designer before you attempt to modify Maintenance Plans created with the Wizard. With this knowledge acquired, you'll find modifying existing plans very easy.

A Maintenance Plan, created using either the Maintenance Plan Wizard or the Maintenance Plan Designer, is implemented "behind the scenes" as a single SQL Server Integration Services

(SSIS) package, executed using one or more SQL Server Agent jobs. While it is possible to manually modify a Maintenance Plan by modifying its SQL Server Agent job, it is not recommended, as there is a strong likelihood of breaking the plan. Instead, always make your changes to a Maintenance Plan using the Maintenance Plan Designer.

> ***Hacking Maintenance Plans***
>
> *If you have the skills and desire to hack a Maintenance Plan in order to get it to perform differently, then I suggest that you would be happier and better off using T-SQL or PowerShell to perform your database maintenance, rather than the Maintenance Plan Designer or Wizard.*

By way of an example, let's say that we want to modify a Maintenance Plan called **User Databases Maintenance Plan** that we originally created using the Maintenance Plan Wizard. The first step is to open up the plan in the Designer, so right-click on the plan's name and select **Modify**. The Maintenance Plan Designer screen appears.

When you originally created the plan in the Wizard, one of the very first screens offered the option to either create a **Separate schedule for each task** (this is the option I recommended), or to create a **Single schedule for the entire plan or no schedule**.

The Maintenance Plan Designer screen will look slightly different, depending on which option you selected. If you chose the **Separate schedule for each task** option, then a separate subplan will be created for each individual maintenance task in the plan, as shown in Figure 19.15.

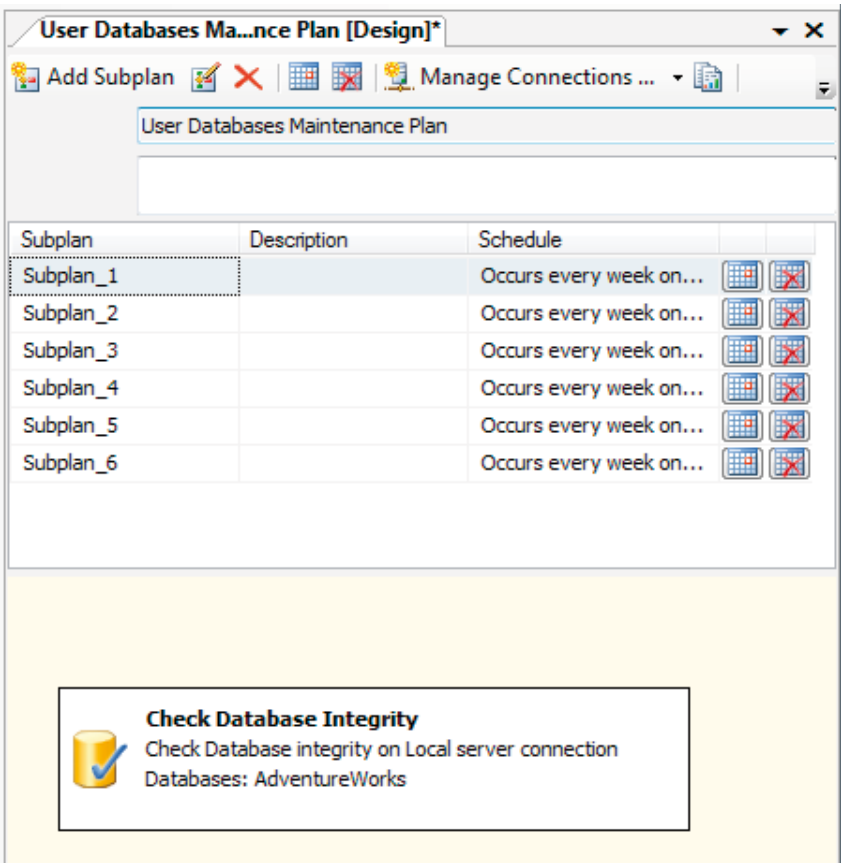

Figure 19.15: In this example, each subplan has a single Maintenance Plan Task.

If, instead, you selected the **Single schedule for the entire plan or no schedule** option in the Wizard, then there will only be a single subplan containing all of the Maintenance Plan tasks, as shown in Figure 19.16. Notice that the precedence arrows reflect the logical ordering you specified for the tasks, within the Wizard.

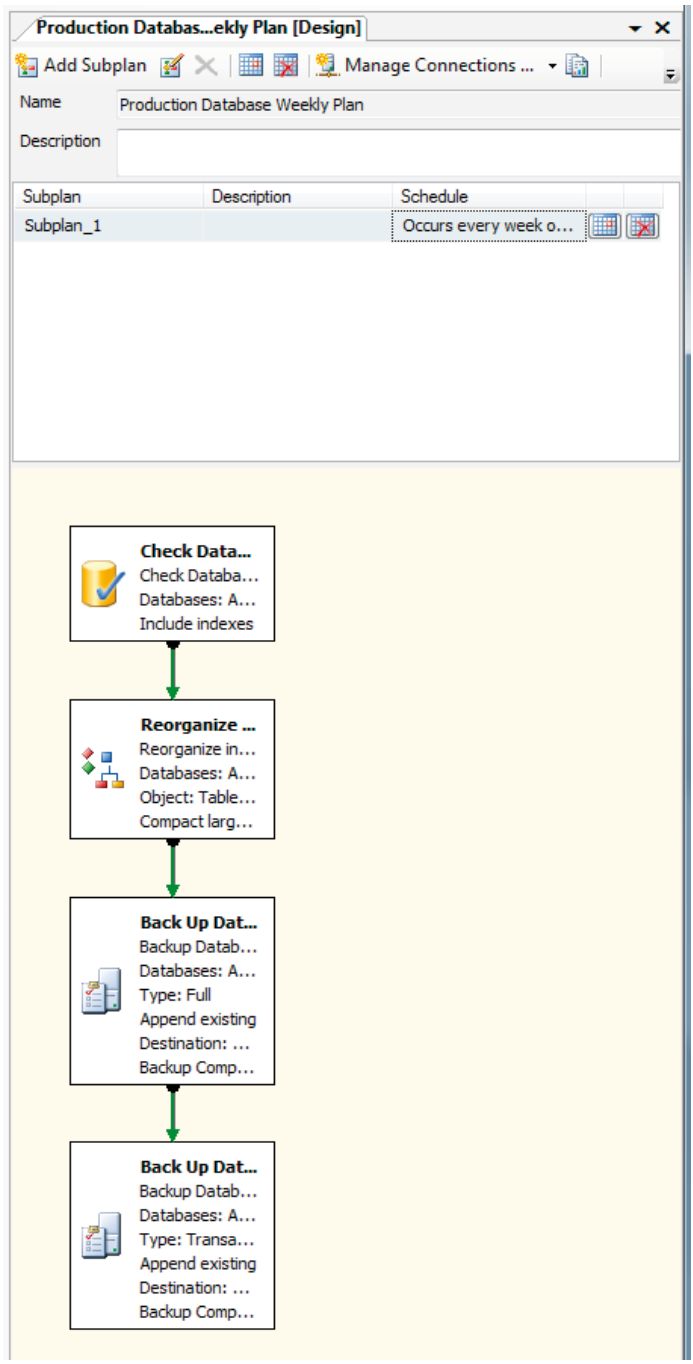

Figure 19.16: Precedence was decided when you ordered the tasks from within the Maintenance Plan Wizard.

Once you have a Wizard-created Maintenance Plan open within Designer, you can modify it in any way you like, just as if you were creating a Maintenance Plan from scratch. Make whatever changes you need, save the plan, test it, and you are ready to reuse it.

As I promised, once you know how to use the Maintenance Plan Designer, modifying Maintenance Plans created with the Wizard is easy.

Summary

Having reached the end of this book, you should now have a good understanding of Maintenance Plans, how to create them using either the Maintenance Plan Wizard or the Maintenance Plan Designer, and the pros and cons of each approach.

The Designer represents a steeper learning curve but the payoff is that it offers a lot more flexibility and power. It is my preferred tool, when creating Maintenance Plans.

What I really want to restate and re-emphasize now is the advice I gave way back in Chapter 1: neither the Maintenance Plan Wizard nor Designer can do all your work for you. The Maintenance Plans you create using these tools offer a very convenient way to perform much of your database maintenance work, but they won't perform other important database maintenance tasks, such as those below.

- Identifying and removing physical file fragmentation.
- Identifying missing, duplicate, or unused indexes.
- Protecting backups so that they are available when needed.
- Verifying that backups are good and can be restored.
- Monitoring performance.
- Monitoring SQL Server and operating system error messages.
- Monitoring remaining disk space.
- And much, much more.

The Wizard and Designer are useful tools for many DBAs, especially when maintaining smaller databases that are not regarded as mission-critical and so have less rigorous maintenance requirements.

If Maintenance Plans meet your needs for a given set of databases, then use them. If they don't meet your needs well, then don't use them. Custom-created T-SQL or PowerShell scripts will offer much more power and flexibility. There is a steeper learning curve attached to creating custom scripts, but it is knowledge that you will be able to use elsewhere as a DBA, and it won't go to waste

About Red Gate

You know those annoying jobs that spoil your day whenever they come up?

Writing out scripts to update your production database, or trawling through code to see why it's running so slow.

Red Gate makes tools to fix those problems for you. Many of our tools are now industry standards. In fact, at the last count, we had over 650,000 users.

But we try to go beyond that. We want to support you and the rest of the SQL Server and .NET communities in any way we can.

First, we publish a library of free books on .NET and SQL Server. You're reading one of them now. You can get dozens more from www.red-gate.com/books

Second, we commission and edit rigorously accurate articles from experts on the front line of application and database development. We publish them in our online journal Simple Talk, which is read by millions of technology professionals each year.

On SQL Server Central, we host the largest SQL Server community in the world. As well as lively forums, it puts out a daily dose of distilled SQL Server know-how through its newsletter, which now has nearly a million subscribers (and counting).

Third, we organize and sponsor events (about 50,000 of you came to them last year), including SQL in the City, a free event for SQL Server users in the US and Europe.

So, if you want more free books and articles, or to get sponsorship, or to try some tools that make your life easier, then head over to www.red-gate.com