SQL Server Crib Sheet Compendium

Amirthalingam Prasanna
Grant Fritchey
Phil Factor
Robert Sheldon
Robyn Page

ISBN: 978-1-906434-14-4
Shelving: Development/Computer Science

In association with
redgate®
www.simpletalkpublishing.com

Table of Contents

Introduction

The 'mission statement' for the Simple-Talk Crib Sheet is:

'For the things you need to know, rather than want to know'

As a developer, DBA or manager, you may not *want* to know all about XML, replication or Reporting Services, but if you next project uses one or more of these technologies heavily then the best place to start is from the 'jungle roof'.

Crib Sheets aim to give you the broad view. Each one tackles a key area of database development, administration or deployment and provides both a management view and a technical view of that topic. Each starts with the business reasons that will underpin a certain technology requirement and then moves on to the methods available to implement them.

A Crib Sheet is not about code solutions – see the *Simple-Talk Workbench* series for that – but about providing a good understanding of all the core concepts and terminology that surround a given technology or discipline. The aim is to cover each topic in just enough detail to perform a given function, no more.

This book contains a collection of Simple-Talk Crib Sheets published between 2006 and 2008. It focuses on SQL Server topics, but also covers two .NET issues that are relevant to all SQL Server developers and DBAs:

- SQL Server Security
- SQL Server XML
- SQL Server Reporting Services
- SQL Server Data Warehousing
- SQL Server Database Backups
- SQL Server Performance
- SQL Server Replication
- Entity Framework
- .NET Performance

Authors

Amirthalingam Prasanna

Prasanna is a software engineer, technical author and trainer with over seven years' experience in the software development industry. He is a Microsoft MVP in the Visual developer category, a MCT and a MCPD on enterprise application development. You can read his blog at www.prasanna.ws and e-mail him at feedback@prasanna.ws.

Prasanna contributed Chapters 9 and 10

Grant Fritchey

Grant is a database administrator for a major insurance company. He has 18 years' experience in IT, including time spent in support and development. He has been working with SQL Server since version 6.0 back in 1995. He worked with Sybase for a few years. He has developed in VB, VB.Net, C# and Java. He is currently working on methods for incorporating Agile development techniques into database design and development at his company.

Grant contributed Chapter 7

Phil Factor

Phil Factor (real name withheld to protect the guilty), aka Database Mole, has 20 years of experience with database-intensive applications. Despite having once been shouted at by a furious Bill Gates at an exhibition in the early 1980s, he has remained resolutely anonymous throughout his career.

Phil contributed to Chapters 1, 2, 3, 6 and 8

Robert Sheldon

After being dropped 35 feet from a helicopter and spending the next year recovering, Robert Sheldon left the Colorado Rockies and emergency rescue work to pursue safer and less painful interests – thus his entry into the world of technology. He is now a technical consultant and the author of numerous books, articles and training material related to Microsoft Windows, various relational database management systems, and business intelligence design and implementation. He has also written news stories, feature articles, restaurant reviews, legal summaries and the novel *Dancing the River Lightly*. You can find more information at http://www.rhsheldon.com.

Robert contributed Chapters 4 and 5.

Robyn Page

Robyn Page is a consultant with Enformatica and USP Networks. She is also a well-known actress, being most famous for her role as Katie Williams, barmaid in the Television Series *Family Affairs*.

Robyn contributed to Chapters 1, 2, 3, 6 and 8

Chapter 1: SQL Server Security Crib Sheet

In a production database, any access to data and processes must be restricted to just those who require it. Generally, the DBA will also want to know who did what within the system, at any point in time.

Each production database will have its own security policy set out, agreed, and documented. This follows on logically from the analysis of the value, sensitivity and nature of the data and processes within the application. It should be updated and available for inspection as part of any audit.

SQL Server's security model is designed to give the flexibility to implement a number of different types of security policy, and allow for all the different application architectures currently in use.

Firstly, SQL Server must only have those features enabled that are absolutely necessary. This is easier to do with SQL Server 2005, but possible with all previous releases. One can cause havoc with such features as Web assistant, Ad-hoc remote queries, OLE Automation, xp_CmdShell, and xp_sendmail. It is always best to start with as many features turned off as possible and configure the database for their use as, or when, needed.

Individuals, or applications, require one or more logins, or memberships of a group login, with which to connect to a database. A simple public-facing website may get its data from a database via one Login, whereas an application with a variety of sensitive, financial, or personal data will have a rich hierarchy of connection types. Ideally, each person who uses an application will have an associated Login. This is not always possible or practical.

Someone with a Login giving access to a Server will need a username, or alias, in each database that he needs to reach within that server. He will, in effect, need to be registered as a user of a database. Furthermore, that user needs permission to access the various objects within the database, such as tables, procedures, views and so on, or to execute code that makes structural changes to the database. Typically, this is done by assigning him to a 'Role', which then has the permissions assigned to it. As people come and go, their membership to the Role is assigned or revoked, without having to fiddle with permissions.

A typical application will be used by a number of different Roles of users, the members of each Role having similar requirements – something like HR, Management-reporting, Dispatch, for example. Each Role will require different types of access to the database depending on their function in the organization.

Each Database Server can therefore manage its security at the server and database level. The 'owner' of a particular database has the power of controlling access to his database via the 'Permission system'. Only the System Administrator can override this.

Overview

SQL Server Security has grown and developed in response to the changing architecture of applications, the demands of application developers and the requirement for simplicity for network administration. SQL Server has tried to keep backward compatibility when it has made these changes, so the result can be slightly confusing on first inspection.

Originally SQL Server had its own simple login and password system, which was completely independent of Windows security, and was logically consistent. All groupings of users were done at database level, and there was just one privileged login to administer the system. This made the adding and removal of users from the network more complex, as it required changing the Logins on every server as well as at the NT Domain level. Integrated security was then introduced, with its concepts of domain users and domain groups, thereby solving some of the problems. There were now, however, groups defined at network level and others, now renamed 'Roles', at database level. The Server-based administration rights were then assigned, as special Roles, to Logins. The database 'Owner' rights were also reworked as 'Fixed Database Roles' that could be reassigned to other database users. However, the old 'SA' login and 'DBO' user were kept for backward-compatibility. SQL Server 2005 has introduced more complexity, such as password policies and execution contexts, in order to tighten security.

Authentication – The Login system

Types of authentication

SQL Server authentication allows the DBA to maintain security and auditing control for the database servers independently of the system manager of the underlying operating system.

The downside of SQL Server's own security is that users have to remember their password for accessing the database and use this each time they connect. They have already supplied a password to log into their PCs. These two different passwords and logins may have different lifetimes and generation rules. Also, this type of security, when used for ODBC/ADO etc, always ends up with passwords being stored in unprotected places. Worse, the passwords are transmitted unencrypted over TCP/IP.

Only SQL Server logins can be used over simple TCP/IP. A connection must have a user name and password, which can be checked against entries in the syslogins table (sys.Server_principals in 2005); otherwise it is terminated.

'Integrated security' can only be used if SQL Server is participating in the Windows Network. The advantages are password-encryption, password-aging, domain-wide accounts and windows administration. It is based on an "access token" which contains the user's unique security ID or 'sid', which is then used by the client to gain access to network resources such as SQL Server, without having to supply login credentials again. If a user has an access token, then it means that he has previously passed authentication checks.

SQL Server can use Windows Security, or use both Windows Security and manage its own user logins. The chances are that unless all access to the server is from within an intranet, both will be required.

Logins

SQL Server will create some Logins automatically on installation (such as. SA), but most are subsequently created by the System administrator. A login ID is necessary for access to a database but not sufficient for most purposes. It has to be granted access to the various resources on the server (Server instance in SQL Server 2005). It holds information that is relevant across databases, such as the user's default language.

Before someone with a Login ID (Except for the SA) can access a database he requires a username or Role within the database, and that username/role must be granted statement permissions and Object permissions. This, traditionally, could only be granted or revoked by the SA or DBO (Database owner). In later versions of SQL Server, this can be done by anyone with the appropriate 'Fixed Server Role', thereby allowing SA rights to be given to domain, or local, Groups of users.

Fixed Server Roles

Logins can, where necessary, be assigned to a number of Fixed Server Roles so that the SA can delegate some, or all, of the administration task. These Roles are:

Sysadmin	can perform any activity, and has complete control over all database functions.
serveradmin	can change server configuration parameters and shut down the server.
setupadmin	can add or remove linked servers, manage replication, create, alter or delete extended stored procedures, and execute some system stored procedures, such as sp_serveroption.
Securityadmin	can create and manage server logins and auditing, and read the error logs.
Processadmin	can manage the processes running in SQL Server.
Dbcreator	can create, alter, and resize databases.
Diskadmin	can manage disk files.

One can therefore create logins using either domain or local users, and one can also create logins with Domain or local groups. One can also create logins with UserID/Password combinations for users who are not part of the Windows network. Any of these can be assigned all or some of the administration rights. On installation there will be:

- A local administrator's Group
- A Local Administrator account
- An SA Login
- A Guest Login

The first three will have the SysAdmin Role by default. The Guest login inherits the permissions of the 'Public' database Role and is used only where a login exists but has no access explicitly granted to the database. If you remove 'guest' from the master database, only the sa user could then log in to SQL Server! When users log in to SQL Server, they have access to the master database as the guest user.

UserNames

Usernames are database objects, not server objects. Logins are given access to a database user by associating a username with a login ID. The Username then refers to the login's identity in a particular database. Additionally, all usernames other than SA can be associated with one or more Roles. When a database is created, a DBO (Database Owner) Role is automatically created, which has full privileges inside the database. However, one can create any number of 'user' Roles. A special Guest Role can be enabled if you want anyone who can log in via a login ID to access a particular database. They will then do it via that Guest Role

Database Roles

A Database Role is a collection of database users. Instead of assigning access permissions to users, one can assign them to Roles, comprising a collection of users who have a common set of requirements for accessing the database: This saves a great deal of work and reduces the chance of error.

If you are just using Integrated security, you can sometimes do without Roles. This is because Logins can represent Domain Groups. If the Domain Group fits the grouping of users required in your database, you can create a username for this group and manage the permissions for this user as if it was a Role.

On creating a database, you should ensure that a server login id exists for everyone who will use the database. If necessary, set the default database in their login to be your new database. If necessary, then create a number of Database Roles depending on the different classes of database access you will have. For each Login (which can represent a group of users). You will need to create a Username. Then you can assign each username to a Database Role. You can subsequently assign permissions to your Roles or Users according to your security plan.

As well as this user-defined Database Role – or Group, as it used to be called – there are fixed Database Roles and the Public Database Role.

Fixed Database Roles

There are several fixed, pre-defined Database Roles that allow various aspects of the database administration to be assigned to users. Members of Fixed Database Roles are given specific permissions within each database, specific to that database. Being a member of a Fixed Database Role in one database has no effect on permissions in any other database. These Roles are…

db_owner	allows the user to perform any activity in the database.
db_accessadmin	allows the user to add or remove Windows NT groups, users or SQL Server users in the database.
db_datareader	allows the user to view any data from all user tables in the database.

db_datawriter	allows the user to add, change, or delete data from all user tables in the database.
db_ddladmin	allows the user to make any data definition language commands in the database.
db_securityadmin	allows the user to manage statement and object permissions in the database.
db_backupoperator	allows the user to back up (but not restore) the database.
db_denydatareader	will deny permission to select data in the database.
db_denydatawriter	will deny permission to change data in the database.

To allow a user to add users to the database and manage roles and permissions, the user should be a member of both the db_accessadmin role and the db_securityadmin role.

Some of these Roles are of a rather specialist nature. Of these Database Roles, possibly the most useful are the db_denydatareader and db_denydatawriter role If the application interface consists entirely of views and stored procedures, while maintaining ownership chains and completely avoiding dynamic SQL, then it is possible to assign the db_denydatareader and db_denydatawriter Role for regular users, to prevent their access to the base tables.

Public Database Role

A Public Database Role is created when a database is created. Every database user belongs to the Public Role. The Public Role contains the default access permissions for any user who can access the database. This Database Role cannot be dropped

Application Roles

Application Roles are the SQL Server Roles created to support the security needs of an application. They allow a user to relinquish his user permissions and take on an Application Role. However, they are not easy to use in conjunction with connection pooling.

Authorisation: The Permissions System

The database user has no inherent rights or permissions other than those given to the Public Role. All rights must be explicitly granted or assigned to the user, the user's Roles, or the Public Role. The Permission System determines which Roles or users can access or alter data or database objects. It determines what every Role or user can do within the database. The SA bypasses the permission system, and so has unrestricted access.

Most commonly, permissions are given to use a database object such as a table, or procedure. Such object permissions allow a user, Role, or Windows NT user or group to perform actions against a particular object in a database. These permissions apply only to the specific object named when granting the permission, not to all the other objects contained in the database. Object permissions enable users to give individual user accounts the rights to run specific Transact-SQL statements on an object.

Permissions can be given or revoked for users and Roles. Permissions given directly to users take precedence over permissions assigned to Roles to which the user belongs. When creating a permission system, it is often best to set up the more general permissions first. Start with the Public Role first and then set up the other Roles, finally doing the overrides for individual users where necessary.

The permission system has a hierarchy of users for which permissions are automatically given.

SA

The SA account is actually a Login rather than a database user. The System Administrator is able to perform server-wide tasks. The System Administrator bypasses the entire permission system and can therefore repair any damage done to the permission system. It can also perform tasks that are not specific to a particular database

Only the System Administrator can create a device, Mirror a device, stop a process, shut down SQL Server, Reconfigure SQL Server, perform all DBCC operations or maintain extended stored procedures. Normally, only the SA creates or alters databases, though this permission can be delegated

DBO

A DBO has full permission to do anything inside a database that he owns. By default, the SA becomes the owner of a database that he creates, but ownership can be assigned. There can be only one DBO for each database. Other than the SA, only a DBO can restore a database and transaction log, alter or delete a database, use DBCC commands, impersonate a database user, issue a checkpoint, grant or revoke statement permissions. The DBO user has all the rights that members of the db_owner role have. The dbo is the only database user who can add a user to the db_owner fixed database role. In addition, if a user is the dbo, when he or she creates an object, the owner of the object will be dbo of that object, as one might expect. This is not true for members of the db_owner Fixed Database Role. Unless they qualify their object names with the dbo owner name, the owner's name will be his or her username.

Normally, a db_owner role member can restore a database, but the information on who belongs to the db_owner Role is stored within the database itself. If the database is damaged to the point where this information is lost, only the DBO can restore the database.

If a user is a member of the db_owner Role but not the dbo, he can still be prevented from accessing parts of the database if 'Deny Permissions' has been set. This does not apply to the the dbo, because the dbo bypasses all permissions checks within the database.

Other DBO roles can be assigned to other users, such as creating objects and Backing up a database or transaction log

DBOO

By default, a user who creates an object is the owner of the object. Whoever creates a database object, the DBOO, or Database Object Owner, is granted all permissions on that object.. Every other user is denied access until they are granted permissions. A user who creates a database object is the DBOO of that object. Members of the db_owner and db_ddladmin Fixed Database Roles can create objects as themselves, their usernames being given as owner, or can qualify the object name as being owned by the dbo.

Assigning Permissions

If the database designer has been able to define an interface based on Stored Procedures, or views, then the permission system will be simple, requiring fewer permissions to be set. The Database administrator will have set up users and Roles and will be able to assign 'Execute' permission to just those procedures that are appropriate for that Role or user. As long as the tables accessed, updated or inserted-into by the stored procedure have the same ownership as the stored procedure (unbroken ownership chain), then permission need not be assigned to the tables. A stored procedure can even update a system table as long as the creator of the stored procedure has the requisite permission when the procedure was created, and the database is configured to allow such a thing. Security can be further enhanced by denying all access by application users to the base tables with db_denydatareader and db_denydatawriter .

If the Database administrator is unfortunate enough to be associated with a database which requires direct access to tables or views, then permissions for 'Select', 'Insert', 'Update' and 'delete' access will need to be assigned directly to the tables that hold your data. They will also entail using column-level permissions, which can overly complicate the security administration model.

If you ever need to grant permission on individual columns of a table, it is usually quicker to create a view, and grant permission on the view. This is carried forward to the individual columns of the tables that make up the view.

It is so unusual for 'Statement permissions' to be assigned that it need not be considered here. However, large development projects may involve the assignment and revoking of permissions to create database objects such as tables, views, procedures, functions, rules and defaults.

Object-level permissions can be to:

Select	Select data from a table view or column
Insert	Insert new data into a table or view
Update	Update existing data in a table view or column
Delete	Delete rows from a table
Execute	Execute a stored procedure, or a function
DRI	allows references to tables that are not owned by the user to be set up directly without select permission
View Definition	(SQL Server 2005 only) Allows the viewing of the metadata.

SQL Server 2005 also provides 'Send', 'Receive', 'Take Ownership' and 'View Definition' object-level permissions

Ownership chains

Sometimes a developer will come up against the problem of 'ownership chains'. When a view or stored procedure is used, permissions are only checked for the contributing objects if there is a change of ownership somewhere along the chain. The most common time this happens is when 'Dynamic SQL' is executed by an Execute() or sp_executeSQL and the user executing the procedure has no permission to access the objects involved. This is known as a Broken Ownership chain, because more than one user owns objects in a dependency chain.

User Context

When SQL Server is running, it needs a 'user context' in which to run. This is the user account that SQL Server uses to access resources on the machine and network. When SQL Server is installed, it is set up with the LocalSystem account, which cannot access the domain. This can be changed for a Domain account if required for backing up to a network disk, or for setting up replication. It is a good idea to use an account where the password is set to 'Password never expires'. SQL Executive will need a domain account in order to publish data for replication.

Chapter 2: SQL Server XML Crib Sheet

This crib sheet is written with the modest ambition of providing a brief overview of XML as it now exists in SQL Server, and the reasons for its presence. It is designed to supplement articles such as Beginning SQL Server 2005 XML Programming.

XML has become the accepted way for applications to exchange information. It is an open standard that can be used on all technical platforms and it now underlies a great deal of the inter-process communication in multi-tiered and distributed architectures.

XML is, like HTML, based on SGML. It is a meta-language, used to define new languages. Although it is not really a suitable means of storing information, it is ideal for representing data structures, for providing data with a context, and for communicating it in context

Previous versions of SQL Server relied on delivering data to client applications as proprietary-format 'Recordsets', either using JDBC or ODBC/ ADO/ OLEDB. This limited SQL Server's usefulness on platforms that could not support these technologies. Before XML, data feeds generally relied on 'delimited' ASCII data files, or fixed-format lists that were effective for tabular data, but limited in the information they could provide.

SQL Server has now been enhanced to participate fully in XML-based data services, and allow XML to be processed, stored, indexed, converted, queried and modified on the server. This has made complex application areas, such as data feeds, a great deal easier to implement and has greatly eased the provision of web services based on XML technologies.

XML has continued to develop and spawn new technologies. There are a number of powerful supersets of XML, such as XHTML, RSS, and XAML that have been developed for particular purposes. A number of technologies have been developed for creating, modifying, and transforming XML data. Some of these have been short-lived, but there are signs that the standards are settling down.

XML

Extensible Markup Language (XML) is a simple, flexible, text-based representation of data, originally designed for large-scale electronic publishing. XML is related to HTML, but is data-centric rather than display-centric. It was developed from SGML (ISO 8879) by employees of Sun (notably Jon Bosak) and Microsoft working for W3C, starting in 1996. In XML, as in HTML, tags mark the start of data elements. Tags, at their simplest, are merely the name of the tag enclosed in '<' and '>' chevrons, and the end tag adds a '/'character after the '<', just like HTML. Attributes can be assigned to elements. The opening and closing tag enclose the value of the element. XML Tags do not require values; they can be empty or contain just attributes. XML tag names are case-sensitive but, unlike HTML, are not predefined.. In XML, there are few restrictions on what can be used as a tag-name. They are used to name the element. By tradition, HTML documents can leave out parts of the structure, such as a </p> paragraph ending. This is not true of XML. XML documents must have strict internal consistency and be 'well formed', to remove any chance of ambiguity. To be 'well formed' they must:

- Have a root element
- Have corresponding closing tags to every tag (e.g. <;address></address>)
- Have tags properly nested.
- Have all attributes enclosed in quotes.
- Have all restricted characters ('<', '>', '", '&' and '"') properly 'escaped' by character entities (<, > ' & ").
- Have matching end-Tags, case-insensitive.

A valid XML document is a well-formed document that conforms to the rules and criteria of the data structure being described in the document. An XML document can be validated against the schema provided by a separate XML Schema document, referenced by an attribute in the root element. This also assigns data types and constraints to the data in the document.

XML Support in SQL Server

SQL Server is fundamentally a relational database, conforming where it can to the SQL standards. XML has different standards, so that integration is made more difficult by the fact that the XML data type standards are not entirely the same as the relational data type standards. Mapping the two together is not always straightforward.

XML has considerable attractions for the DBA or Database developer because it provides a way to pass a variety of data structures as parameters, to store them, query and modify them. It also simplifies the process of providing bulk data-feeds. The challenge is to do this without increasing complexity or obscuring the clarity of the relational data-model.

XML's major attraction for the programmer is that it can represent rowset (single table) and hierarchical (multiple-table) data, as well as relatively unstructured information such as text. This makes the creation, manipulation, and 'persisting' of objects far easier. XML can represent a complex Dataset consisting of several tables that are related through primary and foreign keys, in such a way that it can be entirely reconstructed after transmission.

XML documents can represent one or more typed rowsets (XML Information Set or 'Infoset'). To achieve this, a reference to the relevant XML Schema should be contained in every XML document, or fragment, in order to data-type the XML content. SQL Server now provides a schema repository (library) for storing XML schemas, and it will use the appropriate schema to validate and store XML data.

Loading XML

XML documents of any size are best loaded using the XML Bulk Load facility, which now has the ability to insert XML data from a flat file into an XML column. You can insert XML data from a file into base tables in SQL Server using the OPENROWSET table function, using the 'bulk rowset Provider', with an INSERT statement. The data can then be shredded to relational tables by using the xml.nodes function. (OpenXML can also be used. It is retained by SQL Server for compatibility with SQL Server 2000).

Storing XML

XML documents, XML fragments and top-level text nodes can be stored as XML. XML can be used like any other data type, as a table column, variable, parameter or function return-value. However, there are obvious restrictions: although stored as UTF-16, the XML data is encoded and cannot be directly compared with other XML data, neither can it be used as a primary or foreign key. It cannot have a unique constraint and the XML data is stored in a binary format rather than ASCII.

Unlike other data types, the XML data type has its own methods to Create, Read, Update or Delete the elements within the XML document.

XML data can have default values and can be checked by a variation of the RULE, where the validation is encapsulated within a user-defined function.

XML data types can be allocated data by implicit conversion from the various CHAR formats, and TEXT, but no others. There are no implicit conversions from XML data to other formats.

Checking XML (XML Schemas)

To specify the data type for an element or an attribute in an XML document, you use a schema. XML documents are checked against XML Schemas. The XML Schema is a definition of the data structure used within an XML Document. This indicates, for example, whether a value such as "34.78" (which is stored as a text string within the XML) represents a character string, a currency value, or a numeric value.

If, for example, the XML document represents an invoice, the XML Schema describes the relationship between the elements and attributes, and specifies the data types for that invoice.

You can check, or validate, untyped XML, whether used in a column, variable or parameter, by associating it with an XML Schema. Once checked, it becomes 'typed'. This ensures that the data types of the elements and attributes of the XML instance are contained, and defined, in the schema. These names are valid within the particular 'namespace' specified. An XML Schema definition is itself an XML document. These are catalogued in SQL Server as XML Schema collections, and shredded in order to optimise Schema validation. They are tied to specific SQL Schema within a database.

Using typed XML introduces integrity checking and helps the performance of XQuery.

Accessing Data in XML

XML Data type columns can be indexed, and manipulated, using XQuery and XML Data Manipulation Language (XML DML), which adds 'Insert', 'delete' and 'replace' to XQuery.

To make data-access more effective, XML in SQL Server can be indexed. To be indexed, the XML must be a column in a table that already has a primary key. The index can be over the document structure, or for the values of the elements.

The XML data type can be viewed or modified by a number of methods. One can determine whether a node exists, get its value, retrieve it as table-result of a query, or modify its value.

XML can be read by the XML parser into a 'Document Object Model' (DOM, see below) and then accessed programmatically via methods and properties, but it is not really a suitable server-side technology, due to the overhead of parsing the document into the model.

Shredding XML

The process of converting XML data into a format that can be used by a relational database is called 'Shredding", or decomposition. One can either use the NODES method on an XML data type or, from a Document Object Model (DOM), use the OpenXML function. OpenXML is retained in SQL 2005, but the NODES method is generally preferable because of its simplicity and performance.

Converting relational data to XML

XML fragments, or documents, can be produced from SQL Queries against relational tables, using the SELECT . . . For XML syntax. An inline XSD Format schema can be produced, and added to the beginning of the document. This is convenient but not covered by a W3C standard.

Converting XML to other formats

XML documents can be converted into other XML documents, or into formats such as HTML, using XSL Stylesheets (see below). These are themselves XML documents that provide a mixture of commands and text. It is applied to an XML document by processing it via a parser.

Querying XML Documents

XQuery

XQuery, derived in part from SQL, is the dominant standard for querying XML data. It is a declarative, functional query language that operates on instances of the XQuery/XPath Data Model (XDM) to query your XML, using a "tree-like" logical representation of the XML. With XQuery you can run queries against variables and columns of the XML data type using the latter's associated methods.

XQuery has been around for a while. It evolved from an XML query language called Quilt, which in turn was derived from XML Path Language (XPath) version 1.0, SQL, and XQL.

XQuery has similarities with SQL, but is by no means the same. SQL is a more complete language. The SELECT statement is similar to XQuery's language, but XQuery has to deal with a more complex data model.

The XQuery specification currently contains syntax and semantics for querying, but not for modifying XML documents, though these are effected by extensions to XQuery, collectively called the XML Data Manipulation Language (XML DML). This allows you to modify the contents of the XML document. With XML DML one can insert child or sibling nodes into a document, delete one or more nodes, or replace values in nodes.

Microsoft thoughtfully provided extensions that allow T-SQL variables and columns to be used to bind relational data inside XML data. Server 2005 adds three keywords: insert, update, and delete. Each of these is used within the modify() method of the XML data type.

The XDM that XQuery uses is unlike the Document Object Model (DOM). Each branch (or "node") of the XDM tree maintains a set of attributes describing the node. In the tree, each node has an XML node type, XDM data type information, node content (string and typed representations), parent/child information, and possibly some other information specific to the node type.

FLWOR

XQuery's FLWOR expressions (For, Let, Where, Order by, and Return) iterates XML nodes using the 'for' clause, limits the results using the 'where' clause, sorts the results using the 'order by' clause, and returns the results via the 'return' clause. These constructs greatly extend the versatility of XQuery, making it comparable to SQL

XPath

XPath was designed to navigate an XML document to retrieve the document's elements and attributes. It also provides basic facilities for manipulation of strings, numbers and Booleans. It represents the document as a tree of nodes, and allows reference to nodes by absolute or relative paths. One can specify criteria for the nodes that are returned in square brackets.

XML Template Queries

An XML template query is an XML document with one or more TSQL or XPath queries embedded in it, allowing you to query an XML document. The results can be transformed with an XSLT stylesheet. Template queries are used in client code to update SQL Server data. They are templates with attributes and elements that specify data requiring updating and how that is to be done.

UpdateGram

An UpdateGram is an XML template that is used to insert, update or delete data in a database. It contains an image of the data before and after the required modification. It is usually transmitted to the server by a client application. Each element usually represents one record in a table. The data is 'mapped' either implicitly or explicitly. One can pass parameters to them.

DiffGram

This is an XML document format that is used to synchronise offline changes in data with a database server. It is very similar to an UpdateGram, but is less complex. It is generally used for 'persisting' the data in data objects.

Transforming XML data

XSL

XSL is a stylesheet language for XML that is used to transform an XML document into a different format. It includes XSLT, and also an XML vocabulary for specifying formatting (XSL-FO). XSL specifies the styling of an XML and describes how an XML document is transformed into another document.

Although the resulting document is often HTML, one can transform an XML document into formats such as Text, CSV, RTF, TeX or Postscript. An application designer would use an XSL stylesheet to turn structured content into a presentable rendition of a layout; he can use XSL to specify how the source content should be styled, laid out, and paginated onto some presentation medium. This may not necessarily be a screen display but might be a hand-held device, a set of printed pages in a catalogue, price-list, directory, report, pamphlet, or book.

XSLT

XSLT (XSL Transformations), a language for transforming XML documents into other XML documents, is an intrinsic part of XSL. XSLT and XSL are often referred to as if they were synonymous. However, XSLis the combination of XSLT and XSL-FO (the XSL Formatting Objects).

The Document Object Model

The Document Object Model (DOM) is a platform- and language-neutral interface to enable programs and scripts to dynamically access and update the content, structure and style of XML documents.

XML represents data in a tree structure. Any parser will try to convert the flat text-stream representation of an XML or HTML document into a structured model. The Document Object model provides a standardized way of accessing data from XML, querying it with XPath/XQuery and manipulating it as an object. This makes it a great deal easier for application languages to read or manipulate the data, using methods and objects

The DOM defines the logical structure of the documents and the way they can be accessed. It provides a programming interface for XML documents

SQL Server's OpenXML function actually uses a DOM, previously created using the sp_xml_prepareDocument stored procedure. This function is a 'shredder' that then provides rowsets from the DOM.

XML Web Services

SQL Server 2005 will support web services based on SOAP. SOAP is a lightweight, stateless, one-way message protocol for exchange of information in a decentralized, distributed environment. SQL Server's support makes it much easier for SQL Server to participate in systems based on Unix, Linux or mobile devices.

XML Web services can be placed in the database tier, making SQL Server an HTTP listener. This provides a new type of data access capability for applications that are centralized around Web services, utilizing the lightweight Web server, HTTPSYS, that is now in the operating system, without Internet Information Services (IIS). SOAP can potentially be used with a variety of other protocols other than HTTP but the HTTP-based service is the only one in current use;

SQL Server exposes a Web service interface to allow execution of SQL statements and invocation of functions and procedures. Query results are returned in XML format and can take advantage of the Web services infrastructure of Visual Studio. Web service methods can be called from a .NET application almost like any other method.

A web service is created by:

- Establishing an HTTP endpoint on the SQL Server instance, to configure SQL Server to listen on a particular port for HTTP requests.
- Exposing Stored procedures or user-defined functions as Web Methods
- Creating the WSDL

The web services can include SQL batches of ad-hoc queries separated by semicolons.

Glossary

Character entities	These are certain characters that are represented by multi-character codes, so as not to conflict with the markup.
Infoset	This is an XML document that represents a data structure and is associated with a schema.
Namespace	Namespaces are designed to prevent clashes between data items that have the same name but in different data structures. A 'name', for example, may have different meanings in different part of a data map. Namespaces are generally defined in XML Schemas. Elements in an XML document can be prefixed to attributes. SOAP Namespaces are part of SOAP messages and WSDL files
RSS	An RDF vocabulary used for site summaries.
SGML	The Standard Generalised Markup Language. HTML and XML are applications of SGML
WSDL	Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information
XDM	The Data model used by Xquery to shred XML documents
XDR	XML-Data reduced, a subset of the XML-Data schema method.
XHTML	A language for rendering web pages. It is basically HTML that conforms to general XML rules and can be processed as an XML document.
XML	XML is an acronym for Extensible Markup Language and is a language that is used to describe data and how it should be displayed.
XML Schema	An XML Schema is an XML document that describes a data structure and metadata rather than the data itself
XQuery	XQuery is a query language designed for querying XML data in much the same way that SQL is used, but appropriate to the complex data structures possible in XML documents
XSD	A schema-definition vocabulary, used in XML Schemaa
XSL	A transformation language for XML documents: XSLT. Originally intended to perform complex styling operations, like the generation of tables of contents and indexes, it is now used as a general purpose XML processing language. XSLT is thus widely used for purposes other than XSL, like generating HTML web pages from XML data.

Well-formed XML doc	A well-formed XML document is properly formatted in that the syntax is correct and tags match and nest properly. It does not mean that the data within the document is valid or conforms to the data definition in the relevant XML Schema
XML fragment	This is well-formed XML that does not contain a root element
XQuery	An XML Query language, geared to hierarchical data

.

Happy reading

- XML Support in Microsoft SQL Server 2005
- Beginning SQL Server 2005 XML Programming
- The XML 1.0 standard
- XML 1.1 standard
- The XSL family of recommendations
- HTML Reference
- The W3C website
- The XQuery 1.0/XPath 2.0 Data Model (XDM)

Chapter 3: Reporting Services Crib Sheet

SQL Server Reporting Services (SSRS) aims to provide a more intuitive way of viewing data. It allows business users to create, adapt and share reports based on an abstraction or 'model' of the actual data, so that they can create reports without having to understand the underlying data structures. This data can ultimately come from a variety of different sources, which need not be based on SQL Server, or even relational in nature. It also allows developers many ways of delivering reports from almost any source of data as part of an application.

The reports are interactive. The word 'reporting', in SSRS, does not refer just to static reports but to dynamic, configurable reports that can display hierarchical data with drill-down, filters, sorting, computed columns, and all the other features that analysts have come to expect from Excel. Users can specify the data they are particularly interested in by selecting parameters from lists. The reports can be based on any combination of table, matrix or graph, or can use a customized layout. Reports can be printed out, or exported as files in various standard formats.

SSRS provides a swift, cheap way of delivering to the users all the basic reports that are required from a business application and can provide the basis for customized reports of a more advanced type.

The design of SSRS

The surprising thing about Reporting Services is its open, extensible, architecture. With SSRS, Microsoft has taken pains over a product that has an obvious long-term importance for data handling in .NET.

From a programmer's perspective, the 'Big Idea' behind Reporting Services is to have a standard way of specifying reports. In a way, it is an attempt to do for reports what HTML did for rendering pages. Report Definition Language (RDL) is an XML-based open standard grammar. It was designed to provide a standard way to define reports, to specify how they should appear, their layout and content. It specifies the data source to use and how the user-interaction should work.

In theory, there could be a number of applications to design business reports; several ways of managing them, and a choice of alternative ways of rendering them. All these would work together because of the common RDL format.

SQL Server Reporting Services is the first product to adopt the architecture. It is a combination of report authoring, report management and report delivery. It is not limited to SQL Server data. It can take data from any ODBC source. Reporting Services can use a SQL Server Integration Services package as a data source, thereby benefiting from Analysis Service's multidimensional analysis, hierarchical viewing and data mining. It can just as easily report from OLAP data as relational data. It can also render reports to a number of media including the browser, application window, PDF file, XML, Excel, CSV or TIFF.

The API of SSRS is well enough documented to allow the use of custom data, custom ways of displaying data or special ways of delivering it. Because Microsoft has carefully documented the RDL files and the APIs of the **ReportingServices** namespace, it is reasonably easy to extend the application for special data or security requirements, different data sources, or even the way the reports are rendered. One can of course replace a component such as the report authoring tool with one designed specially for a particular application.

When SSRS is installed, it is set to deliver reports via a 'Report Server' which is installed as an extension to the IIS service on the same server as that on which SQL Server is installed. The actual portal, with its hierarchical menu, report models and security, can be configured either via a browser or from Visual Studio. The browser-based tools are designed more for end-users, whereas the Visual Studio 'Business Intelligence Development Studio' tools are intended for the developer and IT administrator.

The 'Report Server' is by no means the only possible way of delivering reports using Reporting Services, but it is enough to get you started.

So let's look in more detail at the three basic processes that combine to form SQL Server Reporting Services (SSRS): Report Authoring, Report Management and Report Rendering.

The components of SSRS

Report Authoring

The Report Authoring tools produce, as their end-product, RDL files that specify the way that the report will work.

Any application capable of producing an XML file can produce an RDL file, since RDL is merely an XML standard. There is nothing to stop an application from producing an RDL and then using Microsoft's ReportViewer component to render the report.

Hopefully, third-party 'Report Designer' packages will one day appear to take advantage of the applications that are capable of rendering RDL files.

The report designers of SSRS are of two types: 'Report Builder' designed for end users and 'Report Designer' designed for developers.

Report Builder

Report Builder is an 'ad-hoc reporting tool', and designed for IT-savvy users to allow them to specify, modify and share the reports they need. It can be run directly from the report server on any PC with the .NET 2 framework installed. It allows the creation of reports derived from 'report models' that provide a business-oriented model of the data. These reports can then be managed just like any others. The Report Builder allows the users to specify the way data is filtered and sorted, and allows them to change the formulas of calculated columns or to insert new columns. These reports have drill-down features built into them.

Report Designer

Visual Studio has a 'Report Designer' application hosted within Business Intelligence Development Studio. It allows you to define, preview and publish reports to the Report Server you specify, or to embed them into applications. It is a different angle on the task of designing reports to 'Report Builder', intended for the more sophisticated user who understands more of the data and technology. It has a Query Builder, and expression editor and various wizards. The main designer has tabs for the data, layout and preview.

With the embedded Query Designer, you can explore the underlying data and interactively design, and run, a query that specifies the data you want from the data source. The result set from the query is represented by a collection of fields for the dataset. You can also define additional calculated fields. You can create as many datasets as you need to for representing report data. The embedded Layout Designer allows the insertion or alteration of extra computed columns. With the Layout Designer, you can drag fields onto the report layout, and arrange the report data on the report page. It also provides expression builders to allow data to be aggregated even though it has come from several different data locations. It can then be previewed and deployed.

Model Designer

The Model designer in Visual Studio allows you to define, edit and publish 'report models' for Report Builder that are abstractions of the real data. This makes the building of ad-hoc reports easier. These models can be selected and used by Report Builder so that users of the system can construct new reports or change existing reports, working with data that is as close as possible to the business 'objects' that they understand. The model designer allows the programmer to specify the tables or views that can be exposed to the users who can then use the models to design their reports. One can also use it to determine which roles are allowed access to them.

Report Management

There are configuration, monitoring and management tools in SSRS which are provided within the Business Intelligence Development Studio.

Report Manager

Report Manager is a web-based tool designed to ease the management task of connections, schedules, metadata, history and subscriptions. It allows the administrator to categorize reports and control user access. The data models that are subsequently used by the ad-hoc Report Builder tool to translate the data into business entities can be edited in this tool. The report portal , which provides the 'homepage' for the Report Server, can be edited to create or modify the directory hierarchy into which the individual reports are placed. The RDF files can be uploaded to the report server using this tool and placed in their logical position within the hierarchical menu.

One can create or assign the Roles of users that are allowed access the various levels of access to this report. These Roles correspond to previously defined groups in the Active Directory. One can specify whether and how often a report should be generated and email the recipients when the report is ready.

SSRS uses Role-based security to ensure that appropriate access to reports is properly enforced. It controls access to folders, resources and the reports themselves. With SQL Server Standard and Enterprise editions, one can add new Roles, based on Active Directory groups. There are APIs for integrating other security models as well.

Management Studio

The SQL Server Management Studio (SSMS) tool mirrors most of the capabilities of the Report Manager with the addition of instance configuration and scripting. Management Studio itself uses RDL files in order to implement the performance Dashboard so as to get reports on the performance of the server itself. This is easily extended to provide additional reports.

Report Rendering

Viewing Reports on an intranet

When SSRS is installed, it sets up a virtual directory on the local IIS. From there, users with the correct permissions can gain access to whatever reports you choose to deploy. The idea of allowing users to interact with reports and to drill-down into the detail is fundamental to the system, so it is possible to allow users to design their own reports, or use pre-existing ones, and to hyperlink between reports or drill down into data to get more detailed breakdowns. SSRS now provides 'floating headers' for tables that remain at the top of the scrolled list so one can easily tell what is in each column

Report parameters are important in SSRS. If, for example, the users can choose a sales region for a sales report then all possible sales regions for which data exists are displayed for selection in a drop-down list. This information is derived from the data model that forms the basis for the report.

Reports can be viewed via a browser from the report server, from any ASP.NET website and from a Sharepoint portal.

Reports in applications

One is not restricted to browser-based access of SSRS reports. Any .NET application can display such reports easily. The latest version of SSMS, for example, uses reporting services in order to get performance reports.

There are alternatives, such as the Web Browser control or the ReportViewer control.

To use the Web Browser control in an application, all one needs to do is to provide the URL of the report server. The report is then displayed. One can of course launch the browser in a separate window to display the reports. The URL parameters provide precise control over what information is returned. Using the appropriate parameters, not only can you get the report itself for display, you can also access the contents of the Data Source

as XML, the Folder-navigation page, the child items of the report, or resource contents for a report. You can also specify whether it should be rendered on the browser or as an image/XML/Excel file.

The report viewer control, 'ReportViewer', ships with Visual studio 2005 and can be used in any Windows Form or web form surface, just by dragging and dropping. After you assign a report URL and path, the report will appear on the control. You can configure the ReportViewer in a local report-processing mode where the application is responsible for supplying the report data. In local-processing mode, the application can bind a local report to various collection-based objects, including ADO.NET regular or typed datasets.

One can use the Report Server Web Service to gain access to the report management functionality such as content, subscription and data source, as well as all the facilities provided by using a URL request. This allows reporting via any development tool that implements the SOAP methods. This Web Service approach provides a great deal of control over the reporting process and greatly facilitates the integration of Reporting Services into applications, even where the application is hosted in a different operating environment.

SSRS DataSources and Datasets

SSRS Data Sources

Data that is used to provide the Dataset that forms the basis for a report usually comes from SQL Server, or a source for which there is an OLEDB or ODBC provider. It is possible to create the dataset in another application, even a CLR, and bind it to a report. One can access other data sources, such as an ADO.NET dataset, by using a Custom Data Extension (CDE).

Report delivery can be from a Sharepoint site, using the SharePoint Web parts that are included in the SSRS package.

The information contained within a data source definition varies depending on the type of underlying data, but typically includes information such as a server name, a database name, and user credentials.

Data sources can include Microsoft SQL Server, Microsoft SQL Server Analysis Services, ODBC, and OLE DB, Report Server Model, XML, Oracle, SAP NetWeaver Business Intelligence or Hyperion Essbase

A data source can be contained within a report, or it can be shared by several. In the first case, the definition for a report-specific data source is stored within the report itself, whereas for a shared source, the definition is stored as a separate item on the report server. A report can contain one or more data sources, either report-specific or shared.

SSRS DataSets

A Reporting Services DataSet, which is not the same as a .NET dataset, is the metadata that represents the underlying data on a specific data source. It contains a data source definition, a query or stored procedure of the data source and a resulting fields list, also the parameters, if any, of calculated fields as well as the collation. A report can contain one or more datasets, each of which consists of a pointer to a data source, a query, and a collection of fields. These datasets can be used by different data regions on the report, or they can be used to provide dynamic lists of parameters.

The datasets used as the basis for reports can come from a wide variety of sources. Since the examples are mostly queries involving SQL Server base tables, this has given the impression that this is all that can be used. Reports can in fact easily use Stored Procedures to provide the dataset for a report. However, the queries for datasets that fetch the items in the drop-down Parameter lists must be provided too.

Dataset Fields

Each dataset in a report contains a collection of fields. These fields generally refer to database fields and contain a pointer to the database field and a name property but this can be overwritten with a more meaningful name where necessary. Alternatively, these can , be calculated fields, which contain a name and an expression.

Conclusion

When implementing an application, one ignores Reporting Services at one's peril. The benefit to almost any application of implementing standard reports from SSRS is immediate and always impressive to end-users. The impact is far greater than the effort involved. One of us (Phil) suffered intense embarrassment through believing the users of an application when they said that they would never require interactive reports and only wanted strictly defined and cross-checked standard reports in an application. When someone else implemented both Business Intelligence and SSRS, and gave the users the freedom to explore their own data, Phil was left in no doubt about his foolishness in having neglected to do so.

There is always a point when developing an application that the standard fare that can be provided by SSRS is not quite enough for the more advanced reporting requirements. However, it is prudent to make sure that all other reporting up to that point is done via SSRS.

The worst mistake of all is dismissing SQL Server Reporting Services as being just an end-user tool for simple reports. Its architecture is such that it forms the basis of an extremely powerful tool for delivering information to users of an application.

Further Reading….

- SQL Server 2005 Reporting Services
- Technologies: Reporting Services
- SQL Server 2005 Books Online: SQL Server Reporting Services
- Configuring Reporting Services to Use SSIS Package Data
- Introducing Reporting Services Programming (SQL 2000)
- Report Definition Language Specification
- Report Controls in SQL Server 2005 Reporting Services

Chapter 4: SSIS 2008 Crib Sheet

Like most SQL Server 2008 components, SQL Server Integration Services (SSIS) includes a number of new features and enhancements that improve performance and increase developer and administrator productivity. The improvements range from changes to the architecture – in order to better support package development and execution – to the addition of SSIS Designer tasks and components that extend SSIS capabilities and provide more effective data integration.

In this crib sheet, I provide an overview of several of these enhancements and give a brief explanation of how they work. Although this is not an exhaustive list of the changes in SSIS 2008, the information should provide you with a good understanding of the product's more salient new features and help you better understand how these improvements might help your organization.

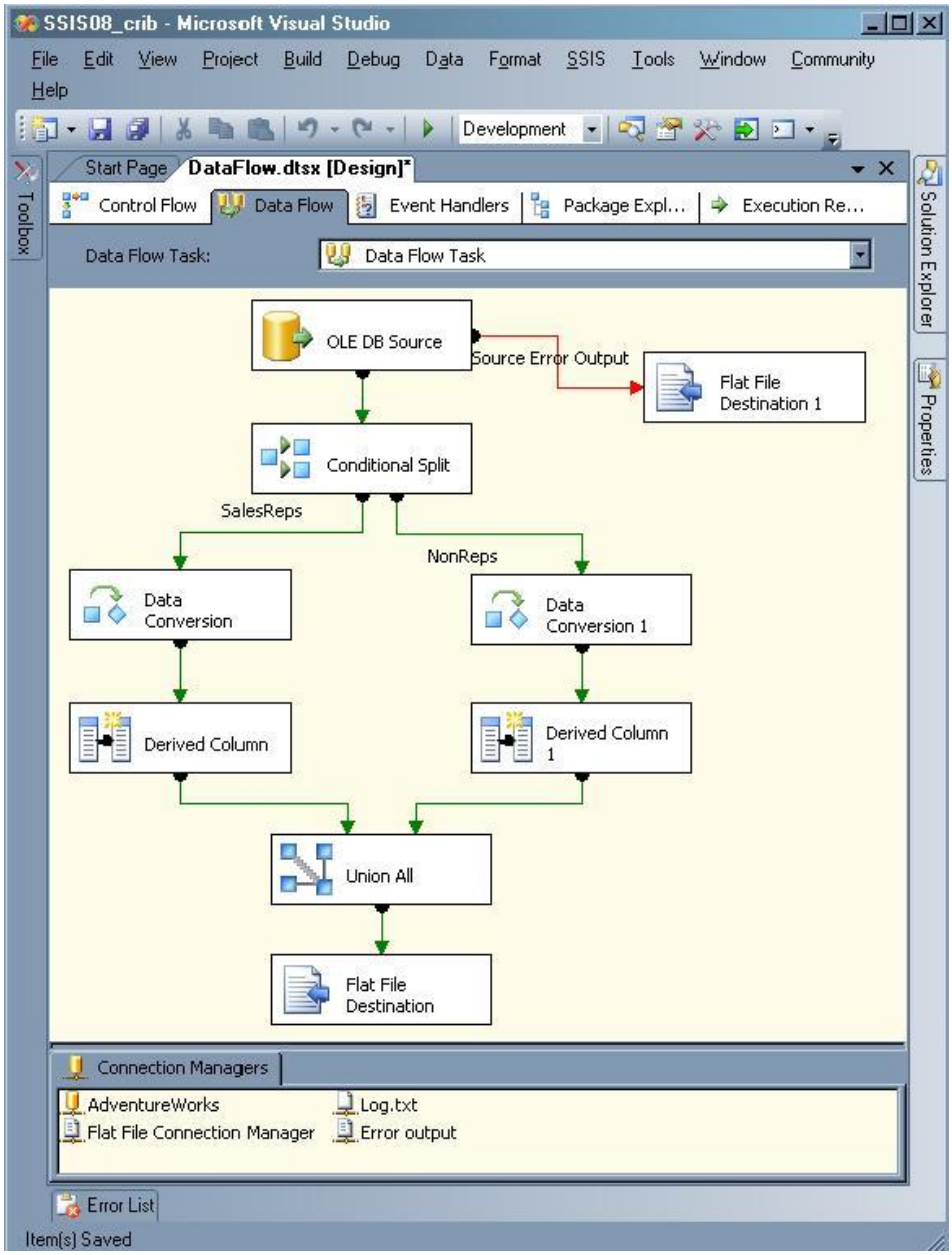
SSIS Architecture

The SQL Server 2008 team has made several important changes to the SSIS architecture, including redesigning the data flow engine, implementing a new scripting environment, and upgrading Business Intelligence Development Studio (BIDS).

Data Flow Engine

In SSIS 2005, the data flow is defined by a set of execution trees that describe the paths through which data flows (via data buffers) from the source to the destination. Each asynchronous component within the data flow creates a new data buffer, which means that a new execution tree is defined. A data buffer is created because an asynchronous component modifies or acts upon the data in such a way that it requires new rows to be created in the data flow. For example, the Union All transformation joins multiple data sets into a single data set. Because the process creates a new data set, it requires a new data buffer which, in turn, means that a new execution tree is defined.

The following figure shows a simple data flow that contains a Union All transformation used to join together two datasets.



Because the Union All transformation is asynchronous – and subsequently generates a new dataset – the data sent to the Flat File destination is assigned to a new buffer. However, the Data Conversion and Derived Column transformations are synchronous, which means that data is passed through a single buffer. Even the Conditional Split transformation is synchronous and outputs data to a single buffer, although there are two outputs.

If you were to log the PipelineExecutionTrees event (available through SSIS logging) when you run this package, the results would include information similar to the following output:

```
begin execution tree 1
   output "OLE DB Source Output" (11)
   input "Conditional Split Input" (83)
   output "SalesReps" (143)
   input "Data Conversion Input" (187)
   output "Data Conversion Output" (188)
   input "Derived Column Input" (148)
   output "Derived Column Output" (149)
   input "Union All Input 1" (257)
```

```
    output "Derived Column Error Output" (150)
    output "Data Conversion Error Output" (189)
    output "NonReps" (169)
    input "Data Conversion Input" (219)
    output "Data Conversion Output" (220)
    input "Derived Column Input" (236)
    output "Derived Column Output" (237)
    input "Union All Input 2" (278)
    output "Derived Column Error Output" (238)
    output "Data Conversion Error Output" (221)
    output "Conditional Split Default Output" (84)
    output "Conditional Split Error Output" (86)
end execution tree 1

begin execution tree 2
   output "OLE DB Source Error Output" (12)
   input "Flat File Destination Input" (388)
end execution tree 2

begin execution tree 0
   output "Union All Output 1" (258)
   input "Flat File Destination Input" (329)
end execution tree 0
```

As the logged data indicates, the data flow engine defines three execution trees, each of which is associated with a data buffer:

Execution tree 1 All components between the OLE DB source output and the Union All input, including the Conditional Split component.

Execution tree 2 From the OLE DB source error output to the Flat File destination input.

Execution tree 0 From the Union All output to the Flat File destination input.

From this, you can see that most of the work is done in the first execution tree. The issue that this approach raises is that in SSIS 2005, the data flow engine assigns only a single execution thread to each execution tree. (To complicate matters, under some conditions, such as when there are not enough threads, a single thread can be assigned to multiple execution trees.) As a result, even if you're running your package on a powerful multiprocessor machine, a package such as the one above will use only one or two processors. This becomes a critical issue when an execution tree contains numerous synchronous components that must all run on a single thread.

And that's where SSIS 2008 comes in. The data flow can now run multiple components in an execution tree in parallel. If you were to run the same package in SSIS 2008, the PipelineExecutionTrees event output would look quite different:

```
Begin Path 0
   output "Union All Output 1" (258); component "Union All" (256)
   input "Flat File Destination Input" (329); component "Flat File Destination" (328)
End Path 0

Begin Path 1
   output "OLE DB Source Output" (11); component "OLE DB Source" (1)
   input "Conditional Split Input" (83); component "Conditional Split" (82)
   Begin Subpath 0
      output "SalesReps" (143); component "Conditional Split" (82)
      input "Data Conversion Input" (187); component "Data Conversion" (186)
      output "Data Conversion Output" (188); component "Data Conversion" (186)
      input "Derived Column Input" (148); component "Derived Column" (147)
      output "Derived Column Output" (149); component "Derived Column" (147)
      input "Union All Input 1" (257); component "Union All" (256)
   End Subpath 0
   Begin Subpath 1
```

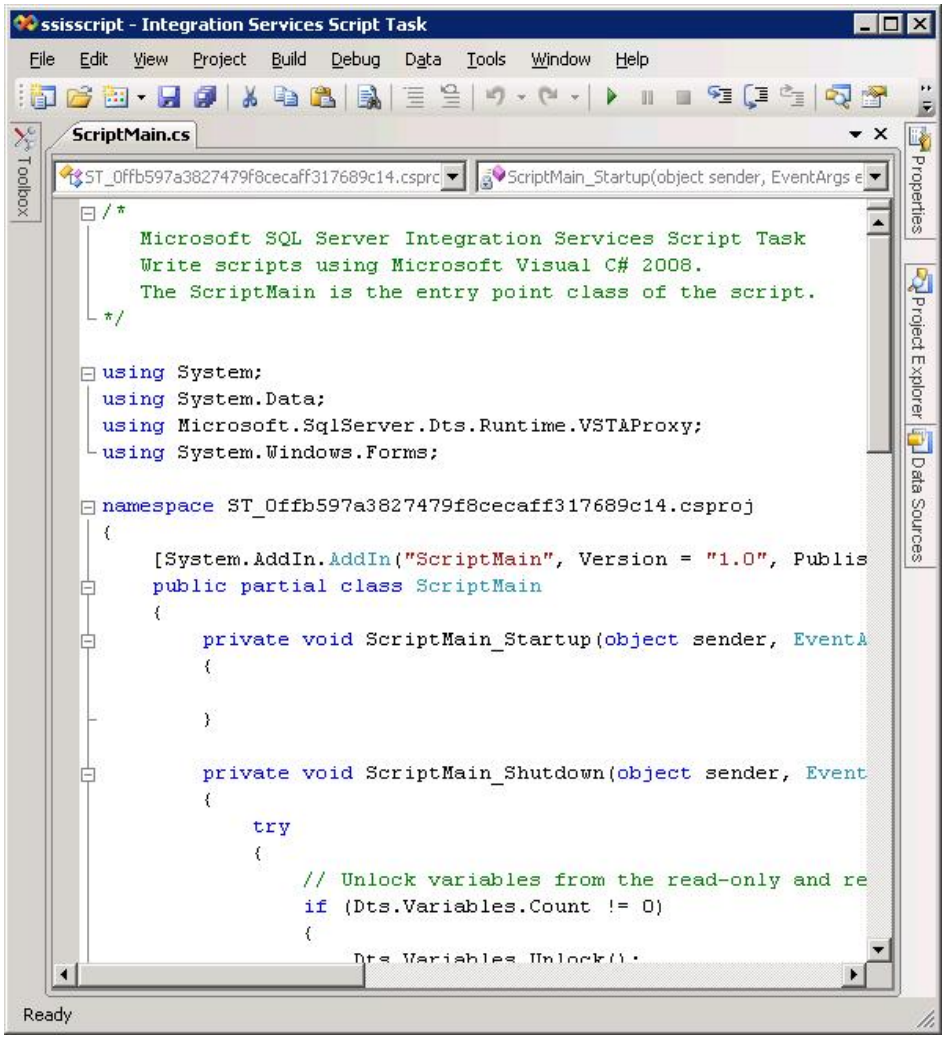
```
      output "NonReps" (169); component "Conditional Split" (82)
      input "Data Conversion Input" (219); component "Data Conversion 1" (218)
      output "Data Conversion Output" (220); component "Data Conversion 1" (218)
      input "Derived Column Input" (236); component "Derived Column 1" (235)
      output "Derived Column Output" (237); component "Derived Column 1" (235)
      input "Union All Input 2" (278); component "Union All" (256)
   End Subpath 1
End Path 1

Begin Path 2
   output "OLE DB Source Error Output" (12); component "OLE DB Source" (1)
   input "Flat File Destination Input" (388); component "Flat File Destination 1" (387)
End Path 2
```

The first thing you'll notice is that the execution trees are now referred to as "paths" and that a path can be divided into "subpaths." Path 1, for example, includes two subpaths, which are each launched with the Conditional Split outputs. As a result, each subpath can run in parallel, allowing the data flow engine to take better advantage of multiple processors. For more complex execution trees, the subpaths themselves can be divided into additional subpaths that can all run in parallel. The best part is that SSIS schedules thread allocation automatically, so you don't have to try to introduce parallelism manually into your packages (such as adding unnecessary Union All components to create new buffers). As a result, you should see improvements in performance for those packages you upgrade from SSIS 2005 to 2008 when you run them on high-end, multiprocessor servers.

Scripting Environment

SSIS includes the Script task and Script component, which allow you to write scripts that can be incorporated into the control flow and data flow. The basic function of the Script task and component are relatively the same in SSIS 2005 and SSIS 2008. However, the difference between the two versions is in the scripting environment itself. In SSIS 2005, the Script task and component use Microsoft Visual Studio for Applications (VSA). In SSIS 2008, the Script task and component use Microsoft Visual Studio 2005 Tools for Applications (VSTA).

The following example shows the ScriptMain class for the Script task in SSIS 2008. The first thing you might notice is that the script is written in C#.



In SSIS 2005, you are limited to writing scripts in Visual Basic.NET. However, in SSIS 2008, because the VSTA environment is used, you can write scripts in C# or Visual Basic.NET.

Another advantage to VSTA is that you can now add Web references to your script. (This option is not available in SSIS 2005.) As a result, you can easily access the objects and the methods available to the Web services. VSTA also lets you add managed assemblies to your script at design time and you can add assemblies from any folder on your computer. In general, VSTA makes it easier to reference any .NET assemblies.

If you're upgrading an SSIS 2005 package to SSIS 2008 and the package contains a Script task or component, SSIS makes most of the necessary script-related changes automatically. However, if your script references IDTS*xxx*90 interfaces, you must change those references manually to IDTS*xxx*100. In addition, you must change user-defined type values to inherit from System.MarshalByRefObject if those values are not defined in the mscorlib.dll or Microsoft.SqlServer.VSTAScriptTaskPrx.dll assemblies.

Business Intelligence Development Studio

In SSIS 2005, BIDS is based on Visual Studio 2005, but in SSIS 2008, BIDS is based on Visual Studio 2008. For the most part, you won't see much difference in your development environment. However, the biggest advantage to this change is that you can have BIDS 2005 and BIDS 2008 installed on the same machine, allowing you to edit SSIS 2005 and 2008 packages without having to switch between different environments.

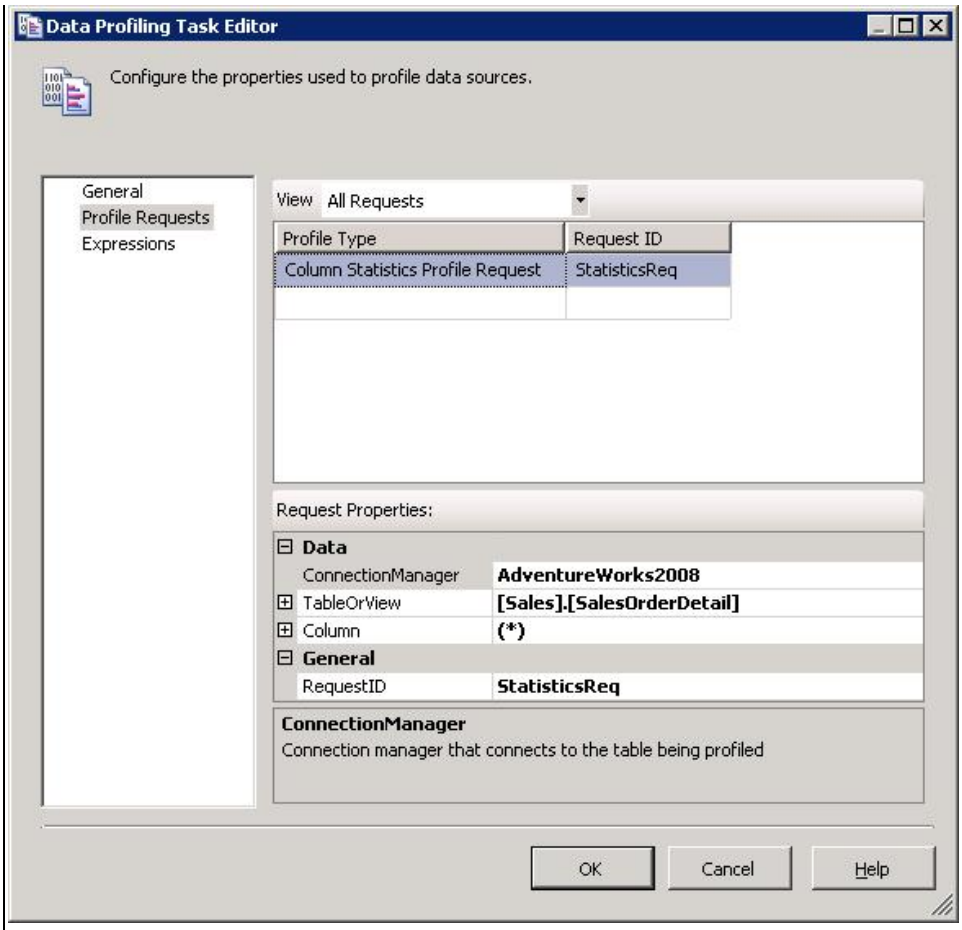
SSIS Designer Tasks and Components

SSIS Designer is the graphical user interface (GUI) in BIDS that lets you add tasks, components, and connection managers to your SSIS package. As part of the changes made to SSIS to improve performance and increase productivity, SSIS Designer now includes the elements necessary to support data profiling, enhanced cached lookups, and ADO.NET connectivity.

Data Profiling

The Data Profiling task, new to SSIS 2008, lets you analyze data in a SQL Server database in order to determine whether any potential problems exist with the data. By using the Data Profiling task, you can generate one or more of the predefined reports (data profiles), and then view those reports with the Data Profile Viewer tool that is available when you install SSIS.

To generate data profile reports, you simply add a Data Profiling task to your control flow and then select one or more profile types in the Data Profiling Task editor (on the Profile Requests page). For example, the following figure shows the Column Statistics profile type.



Although the figure shows only one configured profile type, you can add as many types as necessary, each specific to a data source. The Data profiling task supports eight profile types:

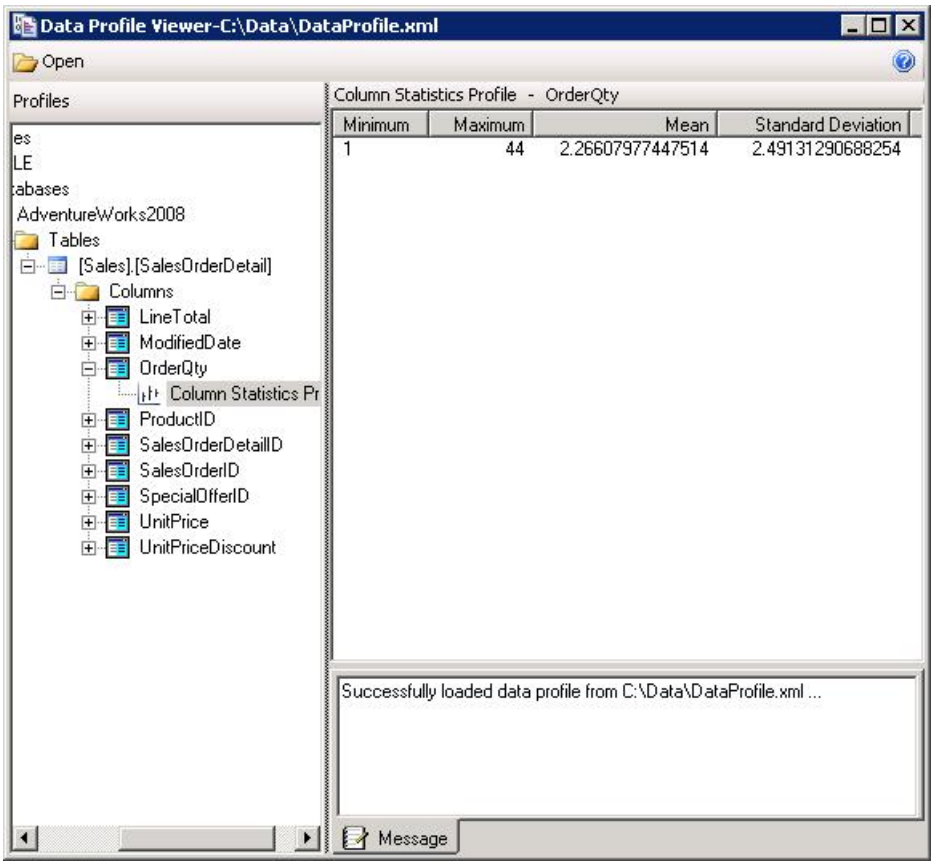
Candidate Key Provides details that let you determine whether one or more columns are suitable to use as a candidate key.

Column Length Distrib. Provides the lengths of distinct values in a string column and the percentage of rows that share each length.

Column Null Ratio	Provides the number and percentage of null values in a column.
Column Pattern	Provides one or more regular expressions that represent the different formats of the values in a column.
Column Statistics	Provides details about the values in a column, such as the minimum and maximum values.
Column Value Distrib.	Provides the distinct values in a column, the number of instances of that value, and the percentage of rows that have that value.
Functional Dependency	Provides details about the extent to which values in one column depend on the values in another column.
Value Inclusion	Provides details that let you determine whether one or more columns are suitable as a foreign key.

For each profile type that you select, you must specify an ADO.NET connection, a table or view, and the columns on which the profile should be based. You must also specify whether to save the profile data to a variable or to an .xml file. Either way, the data is saved in an XML format. If you save the results to a variable, you can then include other logic in your package, such as a Script task, to verify the data. For example, you can create a script that reads the results of a Column Statistics profile and then takes specific actions based on those results.

If you save the data to a file, you can use the Data Profile Viewer to view the data profile that you generated when you ran the Data Profiling task. To use the Data Profile Viewer, you must run the DataProfileViewer.exe utility. By default, the utility is saved to the Program Files\Microsoft SQL Server\100\DTS\Binn folder on the drive where you installed SSIS. After the utility opens, you can open the .xml file from within the utility window. The following figure shows the Column Statistics report generated for the OrderQty column in the Sales.SalesOrderDetail table.

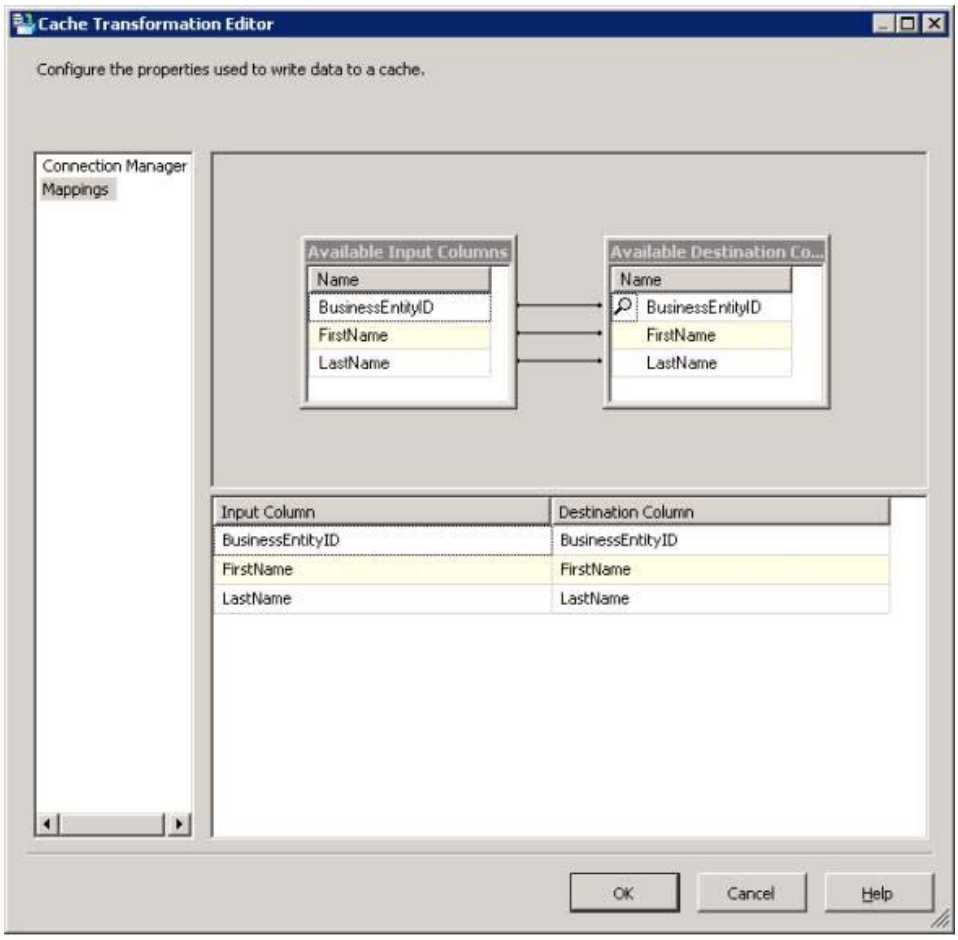


If you specified that multiple reports should be generated, all those reports will be available when you open the file in the Data Profile Viewer. You simply maneuver through the database hierarchy to view the specific data profiles.

Cached Lookups

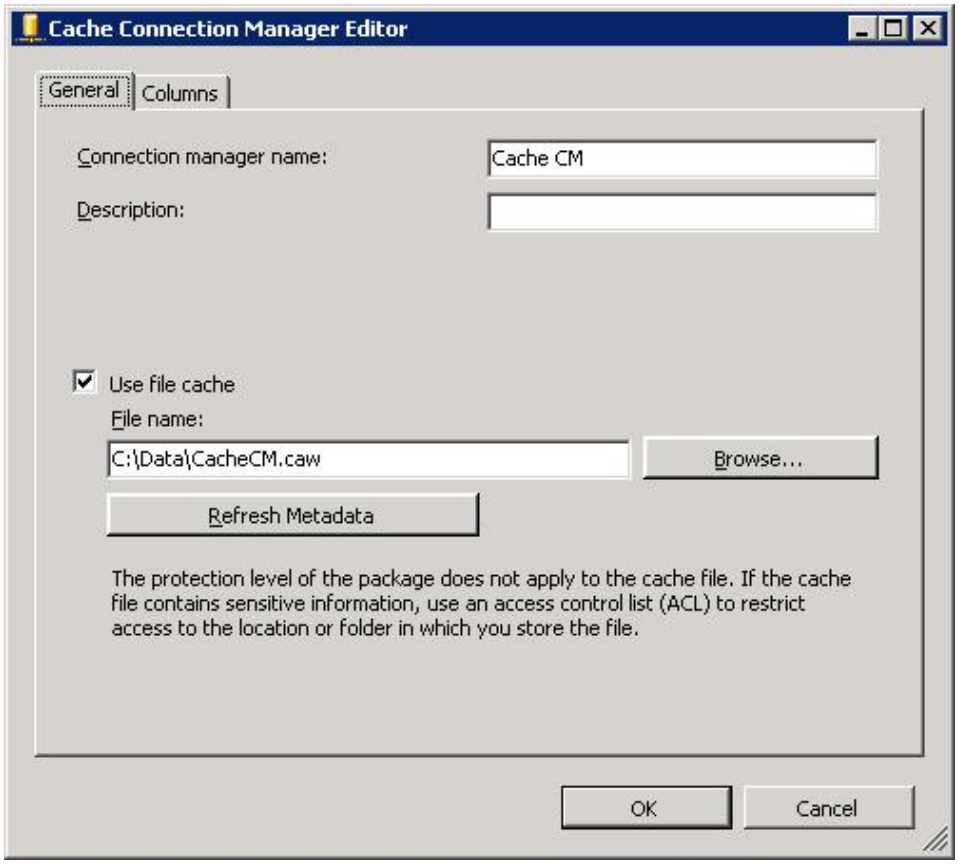
In SSIS 2005, you perform lookup operations in the data flow by using the Lookup transformation to retrieve lookup data from an OLE DB data source. You can, optionally, configure the component to cache the lookup dataset, rather than retrieve the data on a per row basis. In SSIS 2008, your caching options for performing lookup operations have been extended through the Cache transformation and Cache connection manager. By using the new transformation and connection manager, you can cache lookup data from any type of data source (not only an OLE DB source), persist the data to the memory cache or into a cache file on your hard drive, and use the data in multiple data flows or packages.

The primary purpose of the Cache transformation is to persist data through a Cache connection manager. When configuring the transformation, you must – in addition to specifying the connection manager – define the column mappings. The following figure shows the Mappings page of the Cache Transformation Editor.



As you can see, you must map the appropriate input columns to the output columns so the correct lookup data is being cached. In this example, I am caching employee IDs, first names, and last names. I will later use a Lookup transformation to look up the employee names based on the ID.

To support cached lookups, you must – as well as configuring the Cache transformation – configure the Cache connection manager. The following figure shows the General tab of the Connection Manager Editor.



At a minimum, you must provide a name for the connection manager. By default, the lookup data will be stored in the memory cache in the format in which it is received through the data flow pipeline. However, you can instead store the data in a cache (.caw) file by providing a path and file name. You can also modify the data format (data type, length, etc.) on the Columns tab, within the restrictions that govern type conversion in SSIS.

When you use the Cache transformation and connection manager to cache your lookup data, you must perform the caching operation in a package or data flow separate from the data flow that contains the Lookup transformation. In addition, you must ensure that the caching operation runs prior to the package or data flow that contains the Lookup transformation. Also, when you configure the Lookup transformation, be sure to specify full cache mode and use the Cache connection manager you created to support the lookup operation.

ADO.NET

SSIS 2008 now includes the ADO.NET source and destination components. (The ADO.NET source replaces the DataReader source in SSIS 2005; however, SSIS 2008 continues to support the DataReader destination.) The ADO.NET source and destination components function very much like the OLE DB source and destination components. The editors are similar in the way you configure data access, column mappings, error output, and component properties. In addition, because you access data through an ADO.NET connection manager, you can access data through any supported .NET provider, including the ODBC data provider and the .NET providers for OLE DB.

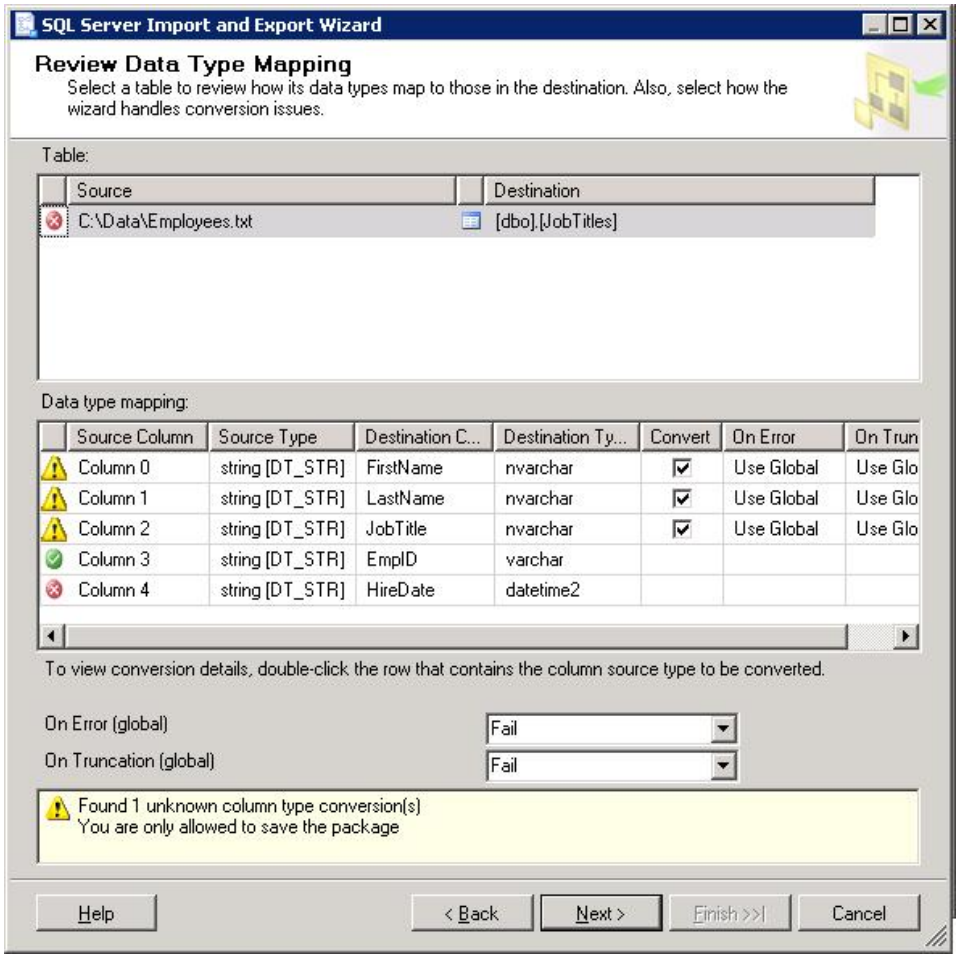
Data Integration

SSIS 2008 has made it easier to work with different types of data by enhancing the SQL Server Import and Export wizard and by adding new SSIS data types to provide better support for data/time data.

SQL Server Import and Export Wizard

When you launch the SQL Server Import and Export wizard in BIDS, the wizard attempts to match the data types of the source data to the destination data by using SSIS types to bridge the data transfer. In SSIS 2005, you had little control over how these SSIS types were mapped to one another. However, in SSIS 2008, a new screen has been added to the wizard to allow you to analyze the mapping so you can address any type mismatch issues that might arise.

The following figure shows the Review Data Type Mapping screen of the SQL Server Import and Export wizard. In this scenario, I am attempting to import data from a text file into a SQL Server database.



The data in the text file is comma-delimited. For this example, you can assume that each row includes the correct number of columns (with the correct data) necessary to match the target table. The target table is based on the following table definition:

```
CREATE TABLE [dbo].[JobTitles](
   [FirstName] [nvarchar](30) NOT NULL,
   [LastName] [nvarchar](30) NOT NULL,
   [JobTitle] [nvarchar](30) NOT NULL,
   [EmpID] [varchar](10) NOT NULL,
   [HireDate] [datetime2](7) NULL
) ON [PRIMARY]
```

If you refer again to the figure above, you'll see that the screen shows how the columns are mapped from the source data to the destination data. Each column is marked with one of the following icons:

Green circle with check mark

The data can be mapped without having to convert the data.

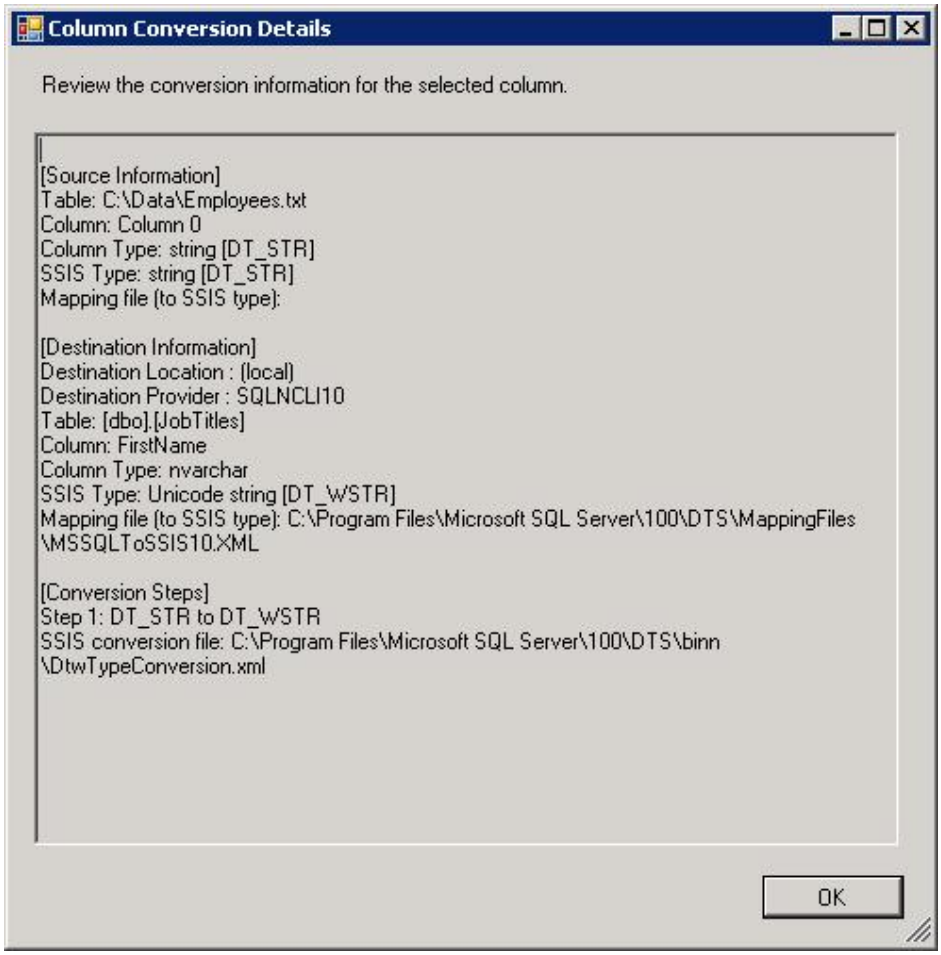
Yellow triangle with exclamation point

The data will be converted based on predefined type mappings. A Data Conversion transformation will be added to the SSIS package that the wizard creates.

Red circle with X

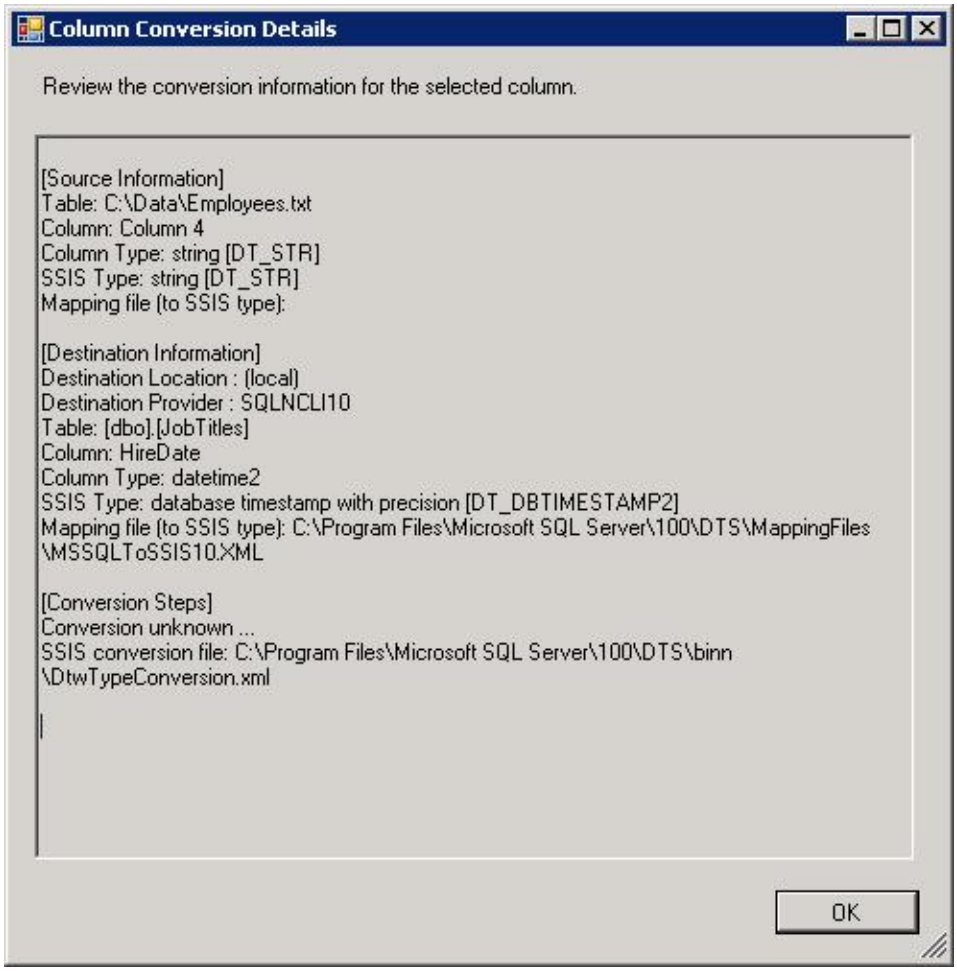
The data cannot be converted. You can save the package but you cannot run it until you address the conversion issue.

As you can see, the first three columns are marked with the yellow warning icons. You can view how the columns will be converted by double-clicking the row for that column information. When you do, a message box similar to the following figure is displayed.



The message box provides details about how the source data will be mapped to an SSIS type, how the destination data will be mapped to an SSIS type, and how the two SSIS types will be mapped. The message box also provides the location and name of the XML files that are used to map the types. Notice that, in this case, the SSIS conversion is from the DT_STR type to the DT_WSTR type – a conversion from a regular string to a Unicode string.

You can also display a message box for the column that shows an error, as shown in the following figure.



As you can see in the last section, the SSIS conversion is unknown. This means that a conversion cannot be mapped between the two SSIS data types that are used to bridge the source and destination data.

To map SSIS data types, SSIS uses the DtwTypeConversion.xml file, which by default is created in the Program Files\Microsoft SQL Server\100\DTS\binn folder on the drive where SSIS is installed. The following XML shows several mappings in the DtwTypeConversion.xml file that are defined by default for the DT_STR data type:

```
<!-- Convert from DT_STR-->
<dtw:ConversionEntry>
  <dtw:SourceType>DT_STR</dtw:SourceType>
  <dtw:DestinationType TypeName="DT_I1">
    <dtw:ConversionStep StepNum="1" ConvertToType="DT_I1"/>
  </dtw:DestinationType>
  <dtw:DestinationType TypeName="DT_I2">
    <dtw:ConversionStep StepNum="1" ConvertToType="DT_I2"/>
  </dtw:DestinationType>
  <dtw:DestinationType TypeName="DT_I4">
    <dtw:ConversionStep StepNum="1" ConvertToType="DT_I4"/>
  </dtw:DestinationType>
```

If a mapping does not exist between two SSIS data types and the data is formatted in such a way that a conversion would be possible, you can add a mapping to the file. For instance, in the example shown in the figures above, the SQL Server Import and Export wizard is trying to map a string in the source data to a DATETIME2 data type in the destination data. (This is the column that is marked as with the error icon.) The first step that the wizard takes is to retrieve the string value as an SSIS DT_STR type. The value then needs to be

converted to an SSIS type consistent with the target column type – DATETIME2. In SSIS, a data type consistent with DATETIME2 is DT_DBTIMESTAMP2. In other words, DT_STR should be converted to DT_DBTIMESTAMP2 in order to bridge the source data to the destination data. However, the DtwTypeConversion.xml file does not contain a DT_STR-to-DT_DBTIMESTAMP2 mapping. If you add this mapping to the file, the wizard will be able to convert the data automatically. When you then run the wizard, you'll see a warning icon rather than an error icon.

Date/Time Data Types

In the previous section, I referenced the DT_DBTIMESTAMP2 data type. This is one of the new date/time data types supported in SSIS 2008. These new types let you work with a wider range of date and time values than those in SSIS 2005. In addition, the new types correspond to several of the new Transact-SQL date/time types supported in SQL Server 2008 as well as those in other relational database management systems (RDBMSs). The following types have been added to SSIS 2008:

DT_DBTIME2 A time value that provides the hour, minute, second, and fractional second up to seven digits, e.g. '14:24:36.5643892'. The DT_DBTIME2 data type corresponds to the new TIME data type in Transact-SQL.

DT_DBTIMESTAMP2 A date and time value that provides the year, month, day, hour, minute, second, and fractional second up to seven digits, e.g. '2008-07-21 14:24:36.5643892'. The DT_DBTIMESTAMP2 data type corresponds to the new DATETIME2 data type in Transact-SQL.

DT_DBTIMESTAMPOFFSET
A date and time value that provides the year, month, day, hour, minute, second, and fractional second up to seven digits, like the DT_DBTIMESTAMP2 data type. However, DT_DBTIMESTAMPOFFSET also includes a time zone offset based on Coordinated Universal Time (UTC), e.g. '2008-07-21 14:24:36.5643892 +12:00'. The DT_DBTIMESTAMPOFFSET data type corresponds to the new DATETIMEOFFSET data type in Transact-SQL.

Moving Forward

In this article, I've tried to provide you with an overview of many of the important new features in SSIS 2008. It is not an exhaustive list, as I mentioned earlier. For example, SSIS 2008 also includes the SSIS Package Upgrade wizard, which lets you easily upgrade SSIS 2005 packages to 2008. In addition, the SSIS executables DTExec and DTUtil now support switches that let you generate memory dumps when a running package encounters an error. You can also take advantage of new features in SQL Server 2008 from within SSIS, such as using Change Data Capture (CDC) to do incremental loads. The more you work with SSIS 2008, the better you'll understand the scope of the improvements that have been made. In the meantime, this article should have provided you with a good foundation in understanding how SSIS 2008 might benefit your organization in order to improve performance and productivity.

Chapter 5: SQL Server Data Warehouse Crib Sheet

One of the primary components in a SQL Server business intelligence (BI) solution is the data warehouse. In a sense, the data warehouse is the glue that holds the system together. The warehouse acts as a central repository for heterogeneous data that is to be used for purposes of analysis and reporting.

Because of the essential role that the data warehouse plays in a BI solution, it's important to understand the fundamental concepts related to data warehousing if you're working with such a solution, even if you're not directly responsible for the data warehouse itself. To this end, the article provides a basic overview of what a data warehouse is and how it fits into a relational database management system (RDBMS) such as SQL Server. The article then describes database modeling concepts and the components that make up the model. It concludes with an overview of how the warehouse is integrated with other components in the SQL Server suite of BI tools.

The purpose of this article is to provide an overview of data warehouse concepts. It is not meant as a recommendation for any specific design. In addition, the article assumes that you have a basic understanding of relational database concepts such as normalization and referential integrity. In addition, the examples used tend to be specific to SQL Server 2005 and 2008, although the underlying principles can apply to any RDBMS.

The Data Warehouse

A data warehouse consolidates, standardizes, and organizes data in order to support business decisions that are made through analysis and reporting. The data might originate in RDBMSs such as SQL Server or Oracle, Excel spreadsheets, CSV files, directory services stores such as Active Directory, or other types of data stores, as is often the case in large enterprise networks. Figure 1 illustrates how heterogeneous data is consolidated into a data warehouse.

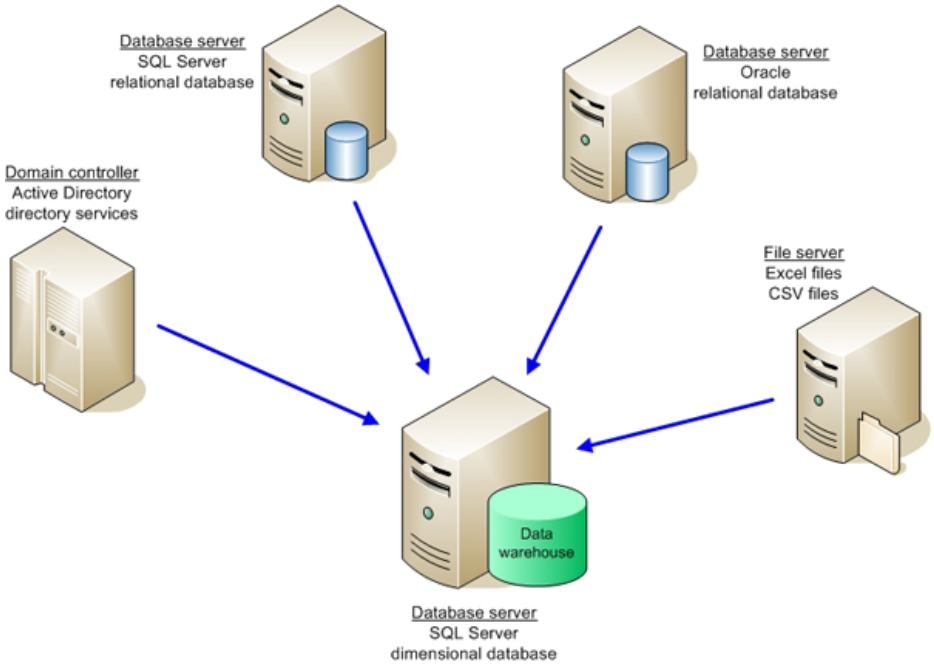


Figure 1: Using a Data Warehouse to Consolidate Heterogeneous Data

The data warehouse must be able to store data from a variety of data sources in a way that lets tools such as SQL Server Analysis Services (SSAS) and SQL Server Reporting Services (SSRS) access the data efficiently. These tools are, in effect, indifferent to the original data sources and are concerned only with the reliability and viability of the data in the warehouse.

A data warehouse is sometimes considered to be a place for archiving data. However, that is not its true purpose. Although historical data is stored in a data warehouse, only the historical range necessary to support analysis and reporting is retained there. For example, if a business rule specifies that the warehouse must maintain two years worth of historical data, older data is offloaded to another system for archiving or is deleted, depending on the specified business requirements.

Data Warehouse vs. Data Mart

A data warehouse is different from a data mart, although the terms are sometimes used interchangeably and there is some debate about exactly what they are and how they differ. It is generally accepted that a data warehouse is associated with enterprise-wide business processes and decisions (and consequently is usually a repository for enterprise-wide data), whereas the data mart tends to focus on a specific business segment of that enterprise. In some cases, a data mart might be considered a subset of the data warehouse, although this is by no means a universal interpretation or practice. For the purposes of this article, we're concerned only with the enterprise-wide repository known as a data warehouse.

Relational Database vs. Dimensional Database

Because SQL Server, like Oracle and MySQL, is a RDBMS, any database stored within that system can be considered, by extension, a relational database. And that's where things can get confusing.

The typical relational database supports online transaction processing (OLTP). For example, an OLTP database might support bank transactions or store sales. The transactions are immediate and the data is current, with regard to the most recent transaction. The database conforms to a relational model for efficient transaction processing and data integrity. In theory, the database design should adhere to the strict rules of normalization which aim, among other things, to ensure that the data is treated as atomic units and that there is minimal amount of redundant data.

A data warehouse, on the other hand, generally conforms to a dimensional model, which is more concerned with query efficiency than issues of normalization. Even though a data warehouse is, strictly speaking, a relational database (because it's stored in a RDBMS), the tables and relationships between those tables are modeled very differently from the tables and relationships defined in the relational database. (The specifics of data warehouse modeling are discussed below.)

Note:
Because of the reasons described above, you might come across documentation that refers to a data warehouse as a relational database. However, for the purposes of this article, I refer to an OLTP database as a relational database and a data warehouse as a dimensional database.

Dimensional Database vs. Multidimensional Database

Another source of confusion at times is the distinction between a data warehouse and an SSAS database. The confusion results because some documentation refers to an SSAS database as a dimensional database. However, unlike the SQL Server database engine, which supports OLTP as well as data warehousing, Analysis Services supports online analytical processing (OLAP), which is designed to quickly process analytical queries. Data in an OLAP database is stored in multidimensional cubes of aggregated data, unlike the typical table/column model found in relational and dimensional databases.

Note:
I've also seen a data warehouse referred to as a staging database when used to support SSAS analysis. Perhaps from an SSAS perspective, this might make sense, especially if all data in the data warehouse is always rolled up into SSAS multidimensional cubes. In this sense, SSAS is seen as the primary database and the warehouse as only supporting that database. Personally, I think such a classification diminishes the essential role that the data warehouse plays in a BI solution, so I would tend to avoid this sort of reference.

OLAP technologies are usually built with dimensionally modeled data warehouses in mind, although products such as SSAS can access data directly from relational database. However, it is generally recommended to use a warehouse to support more efficient queries, properly cleanse the data, ensure data integrity and consistency, and support historical data. The data warehouse also acts as a checkpoint (not unlike a staging database!) for troubleshooting data extraction, transformation, and load (ETL) operations and for auditing the data.

The Data Model

A data warehouse should be structured to support efficient analysis and reporting. As a result, the tables and their relationships must be modeled so that queries to the database are both efficient and fast. For this reason, a dimensional model looks very different from a relational model.

There are basically two types of dimensional models: the star schema and snowflake schema. Often, a data warehouse model follows one schema or the other. However, both schemas are made up of the same two types of tables: facts and dimensions. Fact tables represent a core business process, such as retail sales or banking transactions. Dimension tables store related details about those processes, such as customer data or product data. (Each table type is described in greater detail later in the article.)

The Star Schema

The basic structure of a star schema is a fact table with foreign keys that reference a set of dimensions. Figure 2 illustrates how this structure might look for an organization's sales process.

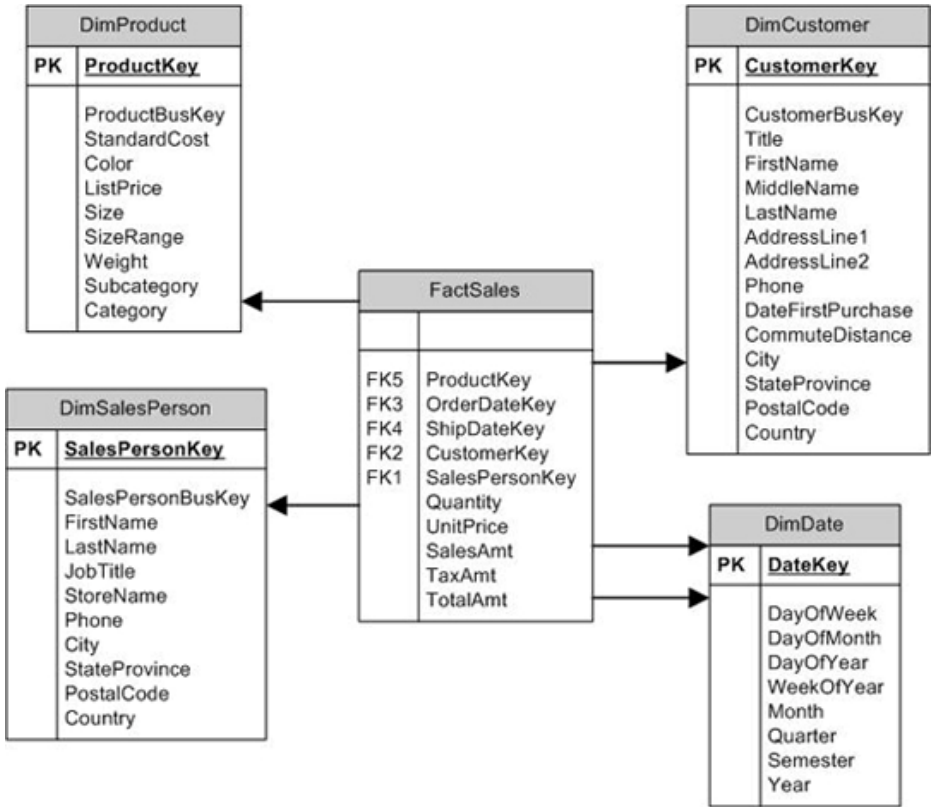


Figure 2: Using a Star Schema for Sales Data

The fact table (**FactSales**) pulls together all information necessary to describe each sale. Some of this data is accessed through foreign key columns that reference dimensions. Other data comes from columns within the table itself, such as **Quantity** and **UnitPrice**. These columns, referred to as *measures,* are normally numeric values that, along with the data in the referenced dimensions, provide a complete picture of each sale.

The dimensions, then, provide details about the functional groups that support each sale. For example, the **DimProduct** dimension includes specific details about each product, such as color and weight. Notice, however, that the dimension also includes the Category and Subcategory columns, which represent the hierarchical nature of the data. (Each category contains a set of subcategories, and each subcategory contains a set of products.) In essence, the dimension in the star schema denormalizes – or flattens out – the data. This means that most dimensions will likely include a fair amount of redundant data, thus violating the rules of normalization. However, this structure provides for more efficient querying because joins tend to be much simpler than those in queries accessing comparable data in a relational database.

Dimensions can also be used by other fact tables. For example, you might have a fact that references the **DimProduct** and **DimDate** dimensions as well as referencing other dimensions specific to that fact. The key is to be sure that the dimension is set up to support both facts so that data is presented consistently in each one.

The Snowflake Schema

You can think of the snowflake schema as an extension of the star schema. The difference is that, in the snowflake schema, dimensional hierarchies are extended (normalized) through multiple tables to avoid some of the redundancy found in a star schema. Figure 3 shows a snowflake schema that stores the same data as the star schema in Figure 2.

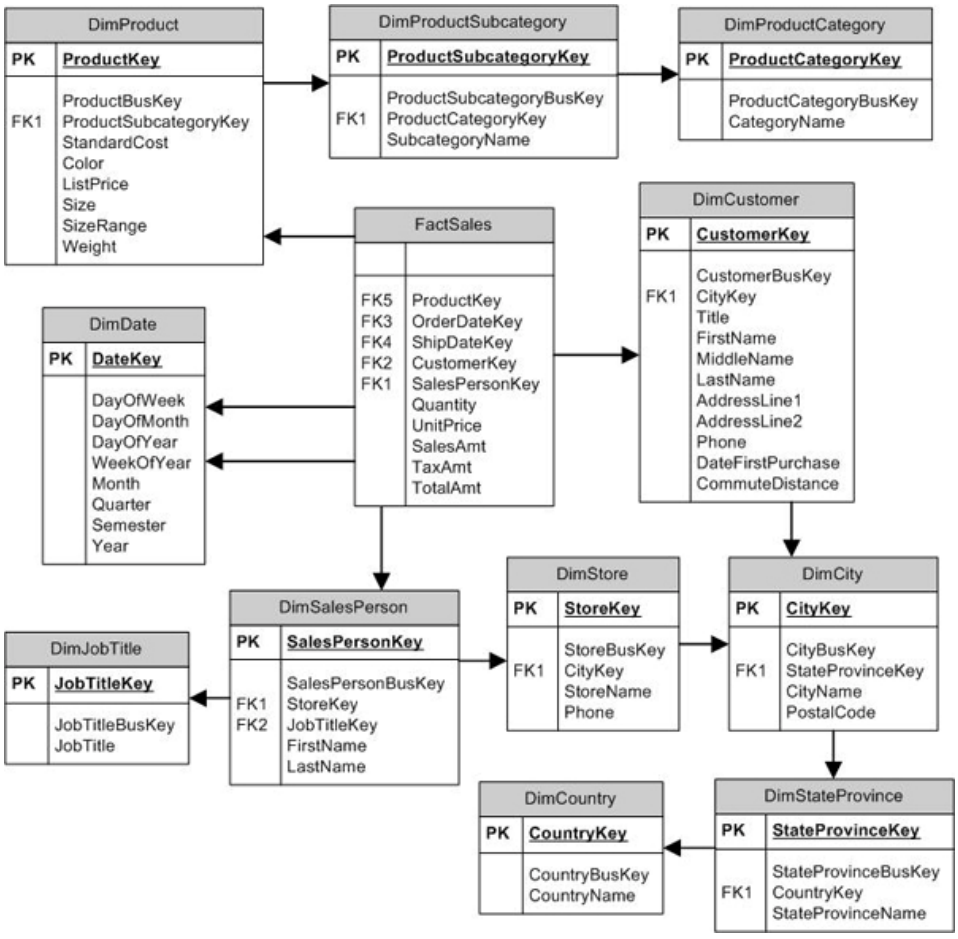


Figure 3: Using a Snowflake Schema for Sales Data

Notice that the dimensional hierarchies are now extended into multiple tables. For example, the **DimProduct** dimension references the **DimProductSubcategory** dimension, and the **DimProductSubcategory** dimension references the **DimProductCategory** dimension. However, the fact table still remains the hub of the schema, with the same foreign key references and measures.

The Star Schema vs. the Snowflake Schema

There is much debate over which schema model is better. Some suggest – in fact, insist – that the star schema with its denormalized structure should always be used because of its support for simpler and better performing queries. However, such a schema makes data updates more complex. The snowflake schema, on the other hand, is easier to update, but it's the queries that are more complex. At the same time, there's the argument that SSAS queries actually perform better against a snowflake schema than a star schema. And the debate goes on.

In many cases, data modelers choose to do a hybrid of both, normalizing those dimensions that support the user the best. The **AdventureWorksDW** and **AdventureWorksDW2008** sample data warehouses take this approach. For example, in the **AdventureWorksDW2008** database, the hierarchy associated with the **DimProduc**t dimension is extended in a way consistent with a snowflake schema, but the DimDate dimension is consistent with the star schema, with its denormalized structure. The important point to keep in mind is that the business requirements must drive the data model – with query performance a prime consideration.

The Fact Table

At the core of the dimensional data model is the fact table. As shown in figures 2 and 3, the fact table acts as the central access point to all data related to a specific business process – in this case, sales. The table is made up of columns that reference dimensions (the foreign keys) and columns that are measures. A fact table supports several types of measures:

- **Additive:** A type of measure by which meaningful information can be extracted by aggregating the data. In the preceding examples, the SalesAmt column is additive because those figures can be aggregated-based on a specified date range, product, salesperson, or customer. For example, you can determine the total sales amount for each customer in the past year.
- **Nonadditive:** A type of measure that does not expose meaningful information through aggregation. In the above examples, the UnitPrice column might be considered nonadditive. For example, totalling the column based on a salesperson's sales in the past year does not produce meaningful data. Ultimately, a measure is considered nonadditive if it cannot be aggregated along any dimensions. (If a measure is additive along some dimensions and not others, it is sometimes referred to as *semiadditive*.)
- **Calculated:** A type of measure in which the value is derived by performing a function on one or more measures. For example, the TotalAmt column is calculated by adding the SalesAmt and TaxAmt values. (These types of columns are sometimes referred to as computed columns.)

If you refer back to figure 2 or 3, you'll see that no primary key has been defined on the fact table. I have seen various recommendations on how to handle the primary key. One recommendation is to not include a primary key, as I've done here. Another recommendation is to create a composite primary key based on the foreign-key columns. A third recommendation is to add an IDENTITY column configured with the INT data type. Finally, another approach is to add columns from the original data source that can act as the primary key. For example, the source data might include an OrderID column. (This is the approach taken by the AdventureWorksDW2008 data warehouse.)

As stated above, the goal of any data warehouse design should be to facilitate efficient and fast queries, while still ensuring data integrity. However, other considerations should include whether it will be necessary to partition the fact table, how much overhead additional indexing (by adding a primary key) will be generated, and whether the indexing actually improves the querying process. As with other database design considerations, it might come down to testing various scenarios to see where you receive your greatest benefits and what causes the greatest liabilities.

The Dimension

A dimension is a set of related data that supports the business processes represented by one or more fact tables. For example, the **DimCustomer** table in the examples shown in figures 2 and 3 contains only customer-related information, such as name, address, and phone number, along with relevant key columns.

The Surrogate Key

Notice that the **DimCustomer** dimension includes the **CustomerKey** column and the **CustomerBusKey** column. Let's start with **CustomerKey.**

The **CustomerKey** column is the dimension's primary key. In most cases, the key will be configured as an **IDENTITY** column with the **INT** data type. In a data warehouse, the primary key is usually a *surrogate key*. This means that the key is generated specifically within the context of the data warehouse, independent of the primary key used in the source systems.

You should use a surrogate for a number of reasons. Two important ones are (i) that the key provides the mechanism necessary for updating certain types of dimension attributes over time and (ii) it removes the dependencies on keys originating in different data sources. For example, if you retrieve customer data from two SQL Server databases, a single customer might be associated with multiple IDs or the same ID might be assigned to multiple customers.

That doesn't mean that the original primary key is discarded. The original key is carried into the dimension and maintained along with the surrogate key. The original key, usually referred to as the business key, lets you map the source data to the dimension. For example, the **CustomerBusKey** column in the **DimCustomer** dimension contains the original customer ID, and the **CustomerKey** column contains the new ID (which is the surrogate key and primary key).

The Date Dimension

A date dimension (such as the **DimDate** dimension in the examples) is treated a little differently from other types of dimension. This type of table usually contains one row for each day for whatever period of time is necessary to support an application. Because a date dimension is used, dates are not tracked within the fact table (except through the foreign key), but in the referenced date dimension instead. This way, not only are the day, month, and year stored, but also such data as the week of the year, quarter, and semester. As a result, this type of information does not have to be calculated within the queries. This is important because date ranges usually play an important role in both analysis and reporting.

Note:
*The **DimDate** dimension in the examples uses only the numerical equivalents to represent day values such as day of week and month. However, you can create a data dimension that also includes the spelled-out name, such as Wednesday and August.*

A fact table can reference a date dimension multiple times. For instance, the **OrderDateKey** and **ShipDayKey** columns both reference the **DateKey** column in **DimDate**. If you also want to track the time of day, your warehouse should include a time dimension that stores the specific time increments, such as hour and minute. Again, a time dimension can be referenced by multiple facts or can be referenced multiple times within the same fact.

Unlike other types of dimensions whose primary key is an integer, a date dimension uses a primary key that represents the date. For example, the primary key for the October 20, 2008 row is 20081020. A time dimension follows the same logic. If the dimension stores hours, minutes, and seconds, each row would represent a second in the day. As a result, the primary key for a time such as 10:30:27 in the morning would be 103027.

The Slowly Changing Dimension

Dimension tables are often classified as slowly changing dimensions (SCDs), that is, the stored data changes relatively slowly, compared to fact tables. For instance, in the previous examples, the FactSales table will receive far more updates than the DimProduct or DimSalesPerson dimensions, whose dimensions normally change very slowly.

The way in which you identify a SCD affects how the dimension is updated during the ETL process. There are three types of slowly changing dimensions:

- **Type 1:** Rows to be updated are overwritten, thus erasing the rows history. For example, if the size of a product changes (which is represented as one row in the DimProduct dimension), the original size is overwritten with the new size and there is no historical record of the original size.
- **Type 2:** Rows to be updated are added as new records to the dimension. The new record is flagged as current, and the original record is flagged as not current. For example, if the product's color changes, there will be two rows in the dimension, one with the original color and one with the new color. The row with the new color is flagged as the current row, and the original row is flagged as not current.
- **Type 3:** Updated data is added as new columns to the dimension. In this case, if the product color changes, a new column is added so that the old and new colors are stored in a single row. In some designs, if the color changes more than once, only the current and original values are stored.

The built-in mechanisms in SQL Server Integration Services (SSIS) implement only Type 1 and Type 2 SCDs. Although you can build an ETL solution to work with Type 3 SCDs, Type 2 SCDs are generally considered more efficient, easier to work with, and provide the best capacity for storing historical data.

When working with SCDs in SSIS, you can use the **Slowly Changing Dimension** transformation to implement the SCD transformation workflow. However, SSIS does not identify SCDs at the dimension level, but rather at the attribute (column) level. Figure 4 shows the **Slowly Changing Dimension Columns** screen of the **Slowly Changing Dimension** wizard.

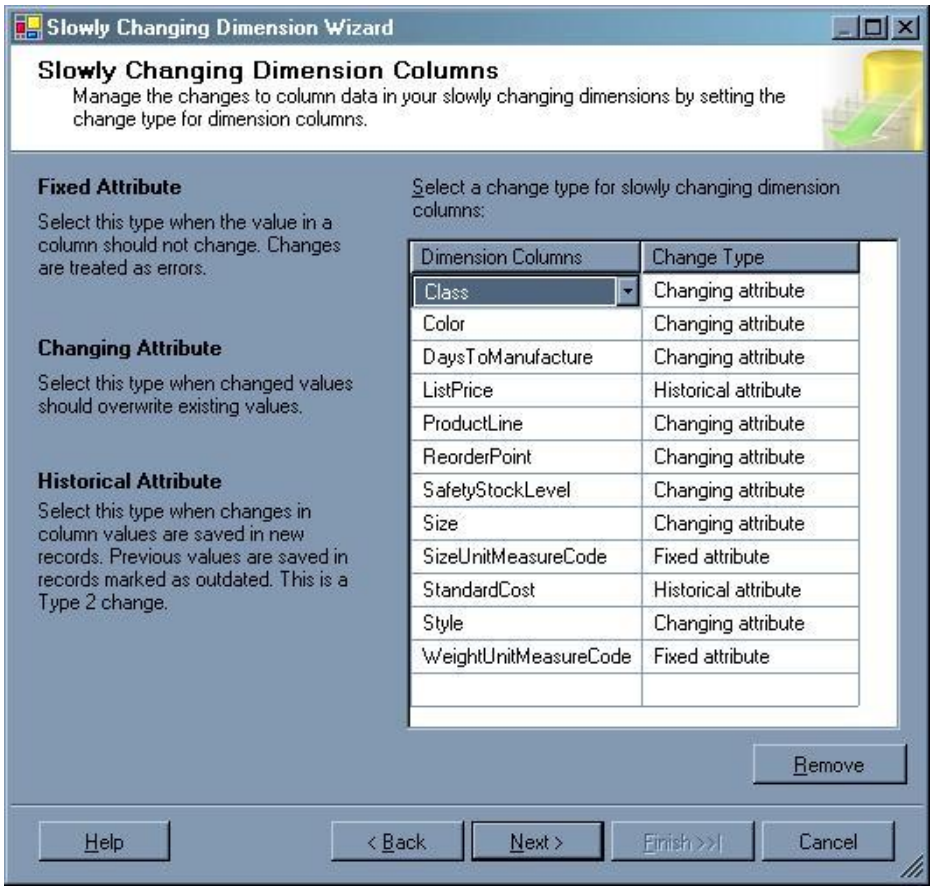


Figure 4: Specifying the Slowly Changing Dimension Columns in SSIS

For each column you can choose whether that column is a **Changing Attribute** (Type 1) or a **Historical Attribute** (Type 2). The wizard also offers a third option – **Fixed Attribute** – in which case, the value of the column can never change.

There is one other consideration when working with SCDs. If a dimension supports Type 2 SCDs, you need some way to mark each row to show if it is the current row (the most recent updated row), and the historical

range of when each row was current. The most effective way to do this is to add the following three columns to any dimension that supports SCD attributes:

- **Start date:** The date/time when the row was inserted into the dimension.
- **End date:** The date/time when the row became no longer current and a new row was inserted to replace this row. If the row is the most current row, this column value is set to null or assigned a maximum date that is out of the present-day range, such as December 31, 9999.
- **Current flag:** A Boolean or other indicator that marks which row within a given set of associated rows is the current row.

Some systems implement only the start date and end date columns, without a flag, and then use the end date to calculate which row is the current row. In fact, the **Slowly Changing Dimension** transformation supports using only the dates or using only a flag. However, implementing all three columns is generally considered to be the most effective strategy when working with SCDs. One approach you can take when using the **Slowing Changing Transformation** wizard is to select the status flag as the indicator and then modify the data flow to incorporate the start and end date updates.

The Data

Although the data warehouse is an essential component in an enterprise-wide BI system, there are indeed other important components. Figure 5 provides an overview of how the SQL Server suite of tools might be implemented in a BI system.

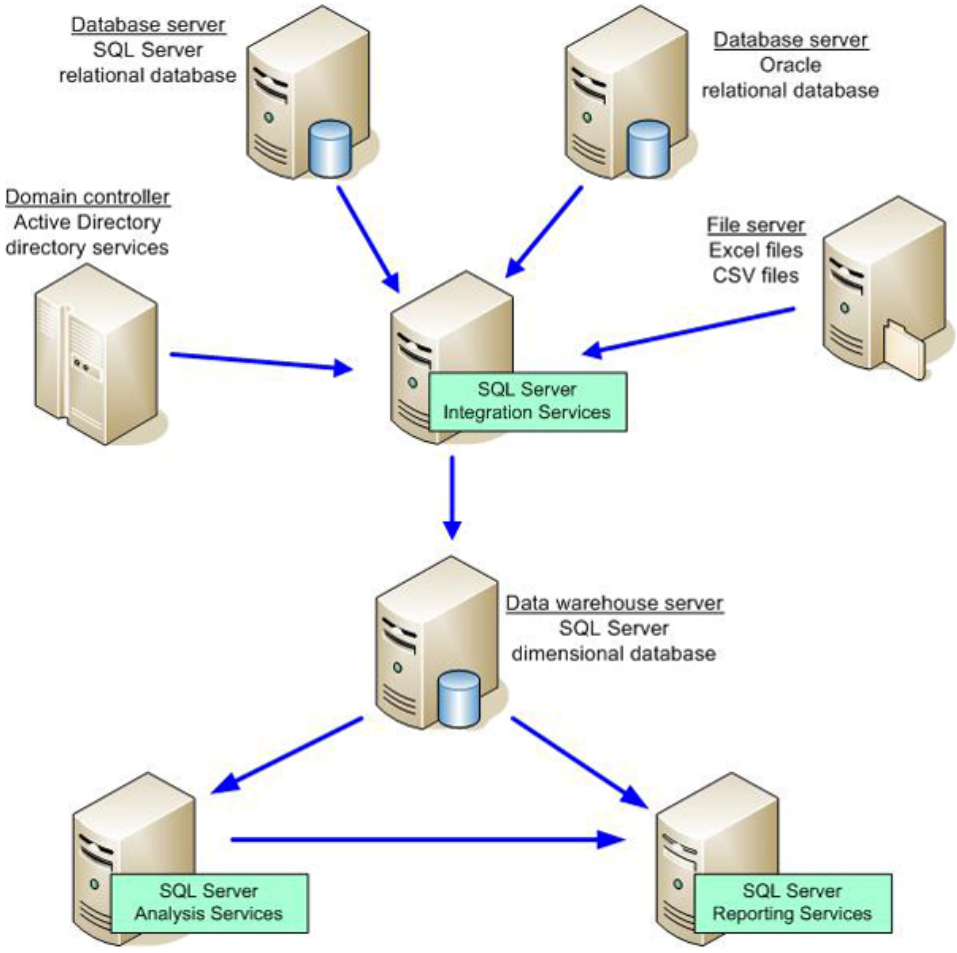


Figure 5: The SQL Server BI Suite

As you can see, SSIS provides the means to retrieve, transform, and load data from the various data sources. The data warehouse itself is indifferent to the source of the data. SSIS does all the work necessary to ensure that the data conforms to the structure of the warehouse, so it is critical that the warehouse is designed to ensure that the reporting and analysis needs are being met. To this end, it is also essential to ensure that the SSIS ETL operation thoroughly cleanses the data and guarantees its consistency and validity. Together, SSIS and the data warehouse form the foundation on which all other BI operations are built.

After the data is loaded into the warehouse, it can then be processed into the SSAS cubes. Note that SSIS, in addition to loading the data warehouse, can also be used to process the cubes as part of the ETL operation.

In addition to supporting multidimensional analysis, SSAS supports data mining in order to identify patterns and trends in the data. SSAS includes a set of predefined data-mining algorithms that help data analyzers perform such tasks as forecasting sales or targeting mailings toward specific customers.

Another component in the SQL Server BI suite is SSRS. You can use SSRS to generate reports based on the data in either the data warehouse or in the SSAS database. (You can also use SSRS to access the data sources directly, but this approach is generally not recommended for an enterprise operation because you want to ensure that the data is cleansed and consistent before including it in reports.) Whether you generate SSRS reports based on warehouse data or SSAS data depends on your business needs. You might choose not to implement SSAS or SSRS, although it is only when these components are all used in orchestration that you realize the full power of the SQL Server BI suite.

Conclusion

Regardless of the components you choose to implement or the business rules you're trying to address in your BI solution, you can see that the data warehouse plays a critical role in that solution. By storing heterogeneous and historical data in a manner that ensures data integrity and supports efficient access to that data, the data warehouse becomes the heart of any BI solution. As a result, the better you understand the fundamental concepts associated with the data warehouse, the more effectively you will understand and be able to work with all your BI operations.

Chapter 6: SQL Server Database Backups Crib Sheet

The simple goal of this crib sheet is to provide a terse but thorough roadmap of all of the important SQL Server backup-related considerations.

General Issues

Database Backup Strategies

Plan this up-front. Any enterprise using databases must maintain a written plan for backing up all data. This plan has to identify the various classes of data held, (e.g. financial, corporate, departmental, personal), the risk of destruction, its value to the enterprise and the general strategy for minimizing the damage or cost that would follow the loss of part, or all, of that data. An agreement must exist for all production system, which defines the maximum tolerable loss of data and the database downtime. The strategy must be tested against various eventualities to verify that a full recovery of the database can be made in all circumstances. The strategy must be available for inspection by external auditors, insurers or assessors.

Why back-up?

Backup is essential because of disk failure, malicious damage, stupidity, fraud, or database corruption, It is also necessary for reversing one or more transactions, copying a database, or archiving a system at a point of time for audit purposes.

What is a backup?

A backup consists of a number of files or 'devices' that together restore the system and user databases to exactly the state of the database system at a particular time. Backups generally consist of a number of full backups and intermediate Log backups. In order to restore a database, the last full backup prior to the point you want to restore to needs to be applied, followed by the log backups prior to that point in time, but subsequent to the full backup. In order to back up a database system, all contributing system databases need to be backed up.

There was a time when databases backups had to be undertaken when the database was offline. This is because any change in the data during the backup process can leave the backup corrupted, as it must necessarily represent the database at a point in time. By the time a backup is completed, an online database would have changed. Now, almost all commercial databases use online backups. This involves writing all completed transactions to disk before copying the database. No transactions after this point are included in the backup. Backing up a database by merely copying its files still must be done with the database offline.

The apparent complexity of a database backup is mostly due to the requirement for online backup, and the occasional need to restore to a particular point in time

When do backups?

This needs to be a conscious decision enshrined in the Service-level agreement for each application. How much time do you want to spend restoring your system? How much data can you afford to lose? How often does the data change in the database? Common practice is that transaction logs need to be backed up "off site" incrementally every few minutes. Complete backups, generally speaking, should be done daily

Where should backups be placed?

Database backups should be ultimately held off site if they are 'financial', or essential to the enterprise. Nowadays this merely means renting internet secure FTP storage. The initial backup is generally done to 'local' disk storage.

What needs backing up?

It is not only the user database that must be backed up. This is also true of the system databases that contribute to the functioning of the server and related servers. The scripts for all database objects, such as stored procedures and triggers, should be held separately in source-control, with a permanent record of all alterations to the objects of the production system.

Who should perform backups?

It shouldn't matter who actually performs the backup, as the procedures will be well documented and will normally be entirely automated.

The DBA nominated by the production manager, or equivalent, needs to have overall responsibility under the terms of the SLA for all 'live' or 'production' databases. The DBA needs to ensure that all production systems have an SLA, that the category of data has been correctly identified and that the SLA is appropriate. He must then devise a backup strategy that meets these requirements. Development systems are done on separate terms and may be supervised by a different chain of responsibility. All developers should ensure that DEV databases are recorded and scheduled for backup. Ad-hoc full backups are sometimes necessary after a non-logged operation, and the DBA must provide all necessary facilities and access rights for the operators who perform such operations, so they can subsequently perform the backup.

Backup media and media rotation

Traditionally, databases used to be 'dumped' onto tape. Tape is slow and not very reliable. Backups are now best written to local disk firstly, and then copied to its ultimate destination. USB drives, MO drives, network drives, NAS or SAN are fine for this, but for databases that are critical to the functioning of the enterprise, a copy must be stored offsite. It is unwise to delete all but the most recent backup, as the integrity of a particular backup cannot be assumed, unless a test restore has been undertaken, followed by a DBCC check. The period for which backups are stored is a matter determined by the backup strategy.

SQL Server issues

Types of Backup

There are several ways one can backup a database. To do an online backup, one would normally choose a combination of complete backup and incremental backup, for which the database must be set to either full or bulk-logged recovery model. However the choice of backup method depends very much on the type of database. Where a database is large and relatively unchanging, then other methods, such as filegroup backups would be more manageable. Generally, system databases must rely only on complete backups. Incremental backups are unique in that they back up the latest transactions of the transaction log, whereas all other types back up the data itself.

Complete or Full backups

Full Backups, or Complete backups, include all the data, system tables and database objects of the database. All backup sets must start with a Full Backup A complete database backup creates a self-sufficient, stand-alone image of the entire database and log at a point in time, and may be restored to any database on any server

A complete backup may be restored for databases regardless of their recovery model.

Complete database backup should be done regularly for all production databases. System databases (including master and MSDB) should have a complete backup if there are any changes performed to the server operating environment such as creating or removing databases, creating and modifying DTS packages or scheduled jobs, configuring security, adding and removing linked servers etc.

Incremental (transaction log) backups

An incremental backup – commonly known as a transaction log backup (the term we use from here in) – stores a backup of all the modifications over a period of time. In other words, a transaction log backup backs up all of the transactions processed by the server for this database since either the previous transaction log backup, or the first complete database backup, whichever is the sooner. This backup may then be used to apply the backed-up changes, to restore the database to a point in time.

Each time the transaction log is backed up all of the committed transactions in the log are removed and written to the backup media. Because of this incremental process, transaction logs are not cumulative. Consider the following backup regime:

10:00 – full backup

10:15 – incremental backup A

10:30 – incremental backup B

10:45 – incremental backup C

…etc…

To restore the database as at 10:45, first restore the full backup at 10:00 and then apply all the incremental backups (A, B and C). Each incremental backup will only restore the transactions over a particular time period (B, for example, stores all transactions between 10.15 and 10.30). The time between incremental backups represents the processing time you are prepared to 'lose' in the restored database if a complete failure occurs.

A non-logged operation done by switching to simple recovery model will mean that the transaction log is out of sync with the database and so a database backup must then be performed.

Transaction log backups are only effective if the database is in full or 'bulk-logged' recovery mode. The Master and MSDB databases are in simple recovery mode and cannot provide effective incremental backups.

A transaction log backup also truncates the inactive portion of the transaction log, unless the database is configured as a Publisher in transactional replication and there are transactions pending propagation to Subscribers.

Transaction log backups must be restored in the correct sequence from oldest to newest.

Differential backups

Differential backups are done in conjunction with a full backup. Whereas transaction log backups can only be used to restore a database if all incremental backups are applied in sequence, differential backups record all the changes (modified extents) since the last full backup and are therefore cumulative. Applying the previous backup regime to differential backups, to restore the database as at 10:45, you would first restore the full backup at 10:00 and then apply differential backup C only.

Differential backups work best for databases that are large and are modified infrequently, such as data warehouses. Usually, this type of database can be restored more quickly from a complete backup followed by a differential backup, than from a series of incremental backups.

Differential backups cannot be used to restore to a point-in-time as they do not use the transaction log. However, one can apply transaction logs after a differential log backup to make a `point in time` recovery.

Differential backups cannot be used for the master database since, when restoring from a differential backup, the first step is to apply the full backup without recovery, which is not allowed for the master database.

File and filegroup backups

Rather than back up the entire database, you can back up individual files and filegroups (a logical grouping of database files) within a database. A file/filegroup backup does not back up the transaction log portion of the database, so cannot be used as the first step to recover a database unless the transaction log is also backed up.

A file/filegroup backup may only be restored to the database it was backed up from, and all transaction log backups, including the 'tail' (log records that have not been backed up), for the database should be restored in their entirety in order to leave it in a consistent state. Because of this, point-in-time recovery cannot be done when restoring file/filegroup backups.

File differential

File differential backups can be combined with file/filegroup backups to back up only the modified extents within an SQL Server database file or filegroup. They are most useful for large, relatively static, databases and are often quicker to restore in cases where only certain file/filegroups are lost. File/filegroup differential backups are cumulative, so in order to restore the files in question you just need the latest differential backup and any transaction log backups performed after the latest file/filegroup differential.

Performing Backups

The following describes a typical backup routine when using T-SQL or the Enterprise manager console in SQL 2000. Options have been tweaked slightly in SQL 2005 Management Studio, but the same basic principles apply.

Backup Type	TSQL	console
Full Backup	BACKUP DATABASE {databasename} TO {device}.	1. Expand the Databases folder, right click on the database in question and select All Tasks | Backup Database. 2. In the Dialog box, provide a name for the backup in the Name text box. 3. Leave the 'Database - complete' option selected. 4. Select either the 'Overwrite existing media' option to overwrite the current backup to existing file or device. Or the "Append" to Media" option to add it to the end. 5. To select a destination for the backup, click the Add button. 6. In the next dialog box, Select an existing file or enter a new file name. Click OK. 7. Click the Options tab (page in SQL 2005). Select the desired options, then either click the OK button to start performing the backup, or check the Schedule check box to schedule this operation for periodic execution.
Transaction log (Incremental)	BACKUP LOG {databasename} TO {device}.	As above, except that, to perform a transaction log backup, select the Transaction log option in step 3. It is unwise to select 'Overwrite Existing

Backup		Media' radio-button. In the Options tab, 'remove inactive entries from transaction log' should be set. The Back up the tail of the Log option should not be set, unless preparing for a restore.
Differential backup	BACKUP DATABASE {databasename}TO {device}WITH DIFFERENTIAL	As above, except that, to perform a differential backup, select the database-differential radio button.
Filegroup Backup	BACKUP DATABASE {databasename} FILE = {filename}, FILEGROUP = {filegroup} TO {device}	As above, except that, to perform a filegroup backup, select the File and filegroup radio button. Click the ellipses button next to this option. In the Specify Filegroups and Files dialog box, select the files/filegroups that you would like to back up, then click OK
Filegroup Differential backup	BACKUP DATABASE {databasename} FILEGROUP = {filegroup}TO {device} WITH DIFFERENTIAL	

System database backups To restore a database server, the system databases will need a backup regime that includes at least the Master database and MSDB

The Master database Must be backed up after creating, altering or dropping a database; altering a data or log file; changing logins; changing configuration options; altering linked servers, remote servers, etc.

MSDB Must be backed-up if the SQL Server Agent is being used to preserve Job Schedule and job history information

Distribution database Needs to be backed up if replication has been configured and the server is a Distributor. There should be a backup after snapshots and, if transactional replication is in place, there should be regular log backups

Publication database This needs to be backed up if replication has been configured and the server is a Distributor, and any a replication setting is changed

Subscription database Needs to be backed up if replication has been configured and the server is a subscriber

Model The model database needs to be backed up only after changes have been made to it.

Recovery models

A SQL Server Database can be set to one of three recovery models. This affects the way that transaction logging is done. The recovery model of the database determines what is possible by way of backup

Simple

If a database is using the simple recovery model, then the database must be restored from the last full backup, because the contents of the database transaction log are truncated each time a checkpoint is issued for the database.

Full

If a database is using the full recovery model, then the database can be restored from the last full backup, followed by the incremental transaction log backups

'Bulk-logged'

A database with 'bulk-logged' recovery will function much like a 'full' one, except that only the effect of bulk operations are recorded in the backups, rather than the transactions involved. These bulk operations include BCP, certain Data Transformation Services (DTS) operations, image and text manipulations, and SELECT INTO. By using a 'bulk-logged' recovery model, the bulk operations are unlogged and therefore much quicker. Point-in-time recovery is not possible if one of these bulk operations have taken place because if a log backup covers any bulk operations, the log backup contains both log records and the data pages that were changed by bulk operations, and this applies to all changes up to the point that the log was backed up.

Set Recovery model	**TSQL**	**console**
Full	ALTER DATABASE {databasename}SET RECOVERY FULL GO	Open the 'Databases' folder. Once the database folder is expanded, right click on the database and select the 'Properties' option. The 'Database Properties' window will open. Click on the 'Options' tab and the recovery model will be listed in the middle of the screen. Click on the drop down box to select the needed recovery model. On the bottom of the screen click 'OK' to save the Recovery Model
Simple	ALTER DATABASE {databasename}SET RECOVERY SIMPLE GO	
Bulk Logged	ALTER DATABASE {databasename}SET RECOVERY BULK_LOGGED GO	

Storage Issues

Replicated Databases

Replicated databases require different strategies for backup which are beyond the scope of this article. Please refer to *Backing up and Restoring Replicated Databases in BOL* for details http://msdn2.microsoft.com/en-us/library/ms151152.aspx

Striped backups

In order to reduce the time taken to perform a backup, SQL Server can write in parallel to up to 32 different devices, grouped together as a 'Media Set'. (A media family is a set of backup units grouped together to make one logical device.) To do the backups, the devices must be defined within a backup device, and must all be of the same type. This is most effective with slow devices such as Tape/DAT drives

Continuation media

If you have to backup directly to tape, rather than to file-then-tape, and the tape fills up during a backup operation, SQL Server prompts for the next tape. (If using the Transact-SQL command, a message is logged to the SQL Server error log to mount the next tape and a retry attempt is made roughly every five minutes to see if a new tape was mounted; if using Enterprise Manager, a dialog box appears.) With disk backups, the backup operation is aborted if the disk drive becomes full.

Standby servers

Where necessary, a backup strategy will involve keeping standby servers ready and waiting, in a state where they can take over the functioning of the primary server without loss of data. The technology to do this usually involves applying the full backups to the spare server and applying the transaction log backups to the standby when the backups are done. Wherever a non-logged operation is performed, the servers need to be resynchronized. You cannot backup or restore a mirror database whilst mirroring is active.

Hot Standby

This only applies to SQL Server 2005, where database mirroring is introduced. A Hot Standby is a server that can switch in automatically on failure of the primary server, and is automatically kept up-to-date (e.g. by database mirroring).

Warm Standby

This represents a server that is automatically updated but has to be manually switched to replace the primary server. This is usually done by log shipping, which normally means creating a new database by restoring a full backup, and then automatically applying the incremental backups to it in order to synchronize it. Alternatively, where the databases are of a reasonable size, an automated synchronization via a third-party synchronization tool provides a useful and reliable alternative.

Cold Standby

Cold Standby servers are manually updated and have to be manually switched to replace the primary server.

Backup History

The history of every SQL Server backup is written to the MSDB database. This can be accessed easily via SQL

Backupfile	Contains one row for each data or log file that is backed up
Backupmediafamily	Contains one row for each media family
Backupmediaset	Contains one row for each backup media set
Backupset	Contains a row for each backup set
Backupfilegroup	Contains one row for each filegroup in a database at the time of backup
Logmarkhistory	Contains one row for each marked transaction that has been committed

suspect_pages	Contains one row per page that failed with an 824 error (with a limit of 1,000 rows)
sysopentapes	Contains one row for each currently open tape device

How Backups should be done

SQL Server Backups are generally automated. The T-SQL for a scheduled backup is placed in the SQL Server Scheduler. However, there are circumstances where a particular backup needs to be started manually –e.g. after a non-logged bulk load. In general, the backups should be administered via the console (either the Enterprise Manager or SSMS according to version) until the regime is settled, tested, and agreed. If required, it can at that point be enshrined in a stored procedure.

When the backup is done via the console, then one can use the automated recovery process. With automated recovery, the process becomes much easier to do, with no issues about which backup, or command flags, to apply and when..

Ideally, the backup procedure will be completely automated by the DBA, and tested in a variety of circumstances by doing various database restores in a number of different circumstances.

Schedule a backup using the Maintenance Plan Wizard

If the concepts explained in this crib sheet are well understood, then the best way of creating a maintenance plan is to use the Maintenance Plan Wizard (Maintenance Plans subsystem Wizard in 2005). However, it will not suit all circumstances and is best used as a guide. It uses sqlmaint instead of native backup commands, and cannot do differential backups.

- For Enterprise Manager in Windows 2000, Right-click the database and select **All Tasks**, then select **Maintenance Plan**
- Click ***<Next>***
- At the first dialog box '***Select Databases dialog box'***, verify that database(s) you want is/are selected by checking them in the list.
- Click ***<Next>*** to get to The dialog box '***Specify data optimisation settings***'
- Select what you wish. Modify the schedule by clicking the Change... button.
- Click ***<Next>*** after selecting fields as necessary to get to '***database integrity check settings***'
- Select the options available to perform a database integrity check.
- Click the ***<Next>*** button to '***Specify database backup plan***' dialog box
- Select the options for your Backup plan
- Click the ***<Next>*** button to proceed to '***Specify database backup directory***' dialog box
- Select the options that suit your backup plan
- Click ***<Next>*** to get to the '***Specify Transaction log backup plan***'
- Select the options that suit your backup plan
- Click ***<Next>*** to get to the '***Specify Transaction log backup directory***'
- Select the options that suit your backup plan
- Click ***<Next>*** to get to '***Specify report generation settings***'
- If selected, a report be generated and should be read to spot any failures that might have occurred.
- click ***<Next>*** to get to '***Specify Maintenance plan history settings***'
- Select to log history records for the execution of this maintenance plan to help troubleshooting. Click the ***<Next>*** '***Maintenance plan summary dialog box***'
- Enter a name for this maintenance plan.
- Click the Finish button to create the maintenance plan.

Third Party Backup tools

The most common problem of the SQL Server backup is that it is written out in an uncompressed form, and can take a large amount of space. This was a good idea in the era of magnetic tapes, when the compression was done in hardware on the device, but it is now an obvious omission. Several third-party products have stepped into the gap to produce products with various other added features such as object-level, and table-level recovery, spreading a backup by 'striping' across multiple files within a directory, log-shipping, and enterprise-wide backup.

Red Gate SQL Backup

Red Gate SQL Backup is able to choose from different compression levels according to the desired trade-off between size of the backup and the speed of the backup. It is able to split backups and encrypt them using different levels of encryption. You can specify the maximum size of the data blocks to be used when SQL Backup writes data to network shares. It includes a Log Shipping wizard to implement log shipping between two servers

Quest Litespeed

Litespeed is one of the oldest third-party backup solutions, with a number of different variations. In the past, it has always majored on its performance and the extent of the compression it can achieve, though independent tests by The Tolly Group in July 2006 to evaluate the disk-to-disk backup performance of SQL Backup Pro 4 and Litespeed show SQL Backup to be faster. Recent 'enterprise' versions have table-level recovery.

Also...

- Netvault Backup
- Hyperbac
- Ultrastor
- Bakbone Netvault
- Idera SQLsafe
- Computer Associates BrightStor Arcserve Backup
- IBM Tivoli Storage Manager
- EMC Dantz Retrospect
- EMC Legato Networker
- Hp OpenView Data Protector
- Veritas NetBackup (Symantec)
- Veritas Backup Exec (Symantec)

Chapter 7: SQL Server Performance Crib Sheet

"*We're not building Amazon.com.*"

Have you heard this statement before, or others like it? This is usually delivered early in a development cycle when someone, probably not the person spewing these words of wisdom, suggests that performance should be taken into account when designing and building the database, or laying out the new server configuration, or writing a trigger. Don't listen to them. Performance is as important to a small system as it is to a large one. A tiny database with 10 users is as important to those 10 users as Amazon.com is to the 10 million users it supports. It's true that worrying about the difference between a 20ms and a 5ms query on the small system may be a waste of time, but there is plenty of work to do before you get down to worrying about that sort of minutiae. The goal of this crib sheet is to provide, in broad strokes, a place to get basic performance information and tuning that applies equally well to SQL Server 2000 or 2005.

Overview

How much performance is enough? Where do you start tuning? When do you stop tuning? Each application being developed will answer these questions in a different way. The important thing is not to establish a single mechanism for answering them. Your goal is to establish best practices and guidelines that will lead to the answers in the right way for the application under consideration.

First, and most important, the SQL Server system itself needs to be configured correctly. It also needs to be running on a correctly configured Windows server. This is the foundation on which the databases will be built. After the server is configured, you need to design and build the database to perform optimally. That's assuming you're building a new database. If you're trying to tune an inherited database, then you'll want to know what a good database looks like. Once the server and database are out of the way, you need to be concerned with the code running against it. This means the views, triggers, functions and, in SQL Server 2005, the CLR code. It doesn't stop there because you need to be sure that the development staff is accessing the database correctly, either in their general use of the database and its code, or in their own use of ADO or whatever other client they might be using to access the system.

In order to address all these concerns you need to understand how to measure and test performance. Once you've measured the performance and found it wanting, you'll need to know what to do about it. After addressing structural changes to the database or changes to the server or refactoring the TSQL code, you'll need to have a mechanism in place to test the results in order to be sure your work is accurate.

After all this, you should have a correctly functioning system that performs and scales well.

Measuring Performance

While setting up the server and designing a database are the foundations on which performance is built, understanding how to measure performance allows you to verify the decisions you've made are the right ones and provides you with direction on future decisions. The two main areas that you'll measure are the performance of the server itself, including both Windows and SQL Server, and the performance of databases and their associated code within SQL Server. To well and truly understand what's happening with your system you'll combine these two sets of measures. There are some tools provided with SQL Server, and whatever flavor of Windows server you're running, that perform these measurements. An entire industry has grown around monitoring and measuring performance of SQL Server and Windows.

Perfmon

Microsoft provides Performance Monitor Counters as a means for keeping an eye on the server. These are accessed through a product called Performance Monitor, commonly referred to as 'perfmon', from the name of

the executable. The counters themselves are grouped into Performance Objects. These vary from the mundane of Physical Disk, Server, Memory, Processor and Cache to the more obscure, like Telephony and WMI Objects, all included as part of a standard server installation. After you install SQL Server an additional list of counters specific to SQL Server Objects to measure how the server is behaving such as SQL Statistics, Buffer Manager, Cache Manager, SQL Server Memory and more are available. It can be quite overwhelming when you consider that each object then lists the available counters. So for example the Buffer Manager object contains 21 counters from 'AWE Lookup Maps/Sec' to 'Total Pages.' Defining a standard set of counters that capture the core information to monitor the system is a must.

Perfmon Counter Set

As an initial list, collecting all the counters for these objects can act as a good baseline for data to make performance tuning decisions:

- Memory
- Network Segment
- Physical Disk
- Processor
- Server
- System
- SQL Server: Access Methods
- SQL Server: Buffer Manager
- SQL Server: General Statistics
- SQL Server: Locks
- SQL Server: SQL Statistics

Using perfmon

When you first open performance. it will have three basic counters for your current machine. Assuming that you're not logged onto the server directly, in order to add counters you need to type in the machine name. This will load the counters for that, so don't panic if you see only the standard set of counters and not SQL server-specific counters. There are two ways of using perfmon:

- You can add a series of counters and observe their behavior through the GUI in real time.
- You can use perfmon's ability to create Counter Logs with Window's Scheduler to record counters for later review and manipulation.

The latter approach is certainly more systematic. Counter Logs are collections of data written to a file. You can run them over a period of time, say during year-end processing if that's when you're expecting high load, or over the course of a two-hour test (more on that below in Performance Testing). Instead of simply watching the counters scroll by and losing the data forever, you'll have a record that allows you to isolate when and where bottlenecks occurred.

A key point to keep in mind is that perfmon has a data collection cycle interval. This means that if you're experiencing high loads, but short transactions, a 30-second sample rate may entirely miss the events you'd most like to record. With this in mind, when not performing a long running system test, a low interval like 10 seconds would not be unreasonable. Remember that the lower the interval, the more data will be collected. Keep this in mind when planning for disk space. The log collection can contain start and stop times or define a length of time for it to run. All these settings are available through the command line, so you can start data collection using third party tools or schedulers.

Supported file formats include binary, comma-delimited and tab-delimited. You can also store the data directly to SQL Server, but for most performance monitoring situations, storing the data to file rather than to SQL Server works well. It means you can run perfmon on a machine other than that which is being monitored, which means that any I/O costs are not incurred by the monitored machine. If you were to use SQL Server – probably to set up some sort of enterprise level of monitoring – you would want to be careful to not use the machine you're

monitoring to also store the data being collected, as this will mask the normal performance of the machine behind the transactions necessary to support perfmon.

One other source of counter information in SQL Server 2005 is the new dynamic management view **sys.dm_os_performance_counters**. These are only a sub-set of the counters available in perfmon, but this subset is immediately available inside queries for whatever monitoring solution you might be trying to put in place.

Evaluating perfmon data

Having collected the data on your server during high production use, or as part of a load test, you need to know how to evaluate these counters. Each set of counters and each individual counter tells a different part of the story. A massive amount of drill-down is required to define all of them. Instead, I'll focus on a very few of the most important counters – ones that can directly indicate the health of your overall system, the Server itself and SQL Server.

Server health

Starting with the server itself, you need to worry about:

- Memory
- Disk I/O
- The Processors

Memory

The most basic memory check is 'Pages/Sec'. According to the MS documentation, this counter shows the:

"*Rate at which pages are read from or written to disk to resolve hard page faults. This counter is a primary indicator of the kinds of faults that cause system-wide delays.*"

A high number here means there is a lot of activity in the memory of your system. What constitutes "a high number" depends on the amount of memory on the machine. My laptop shows spikes of 156 pages/Sec under a light load, whereas one of my production servers can show a spike of over 1,000 under a normal load. Measuring your system over time will allow you to develop the knowledge of what constitutes an abnormal load.

Disk I/O

The core check for disk drives is the length of time that writes (including updates, inserts and deletes) or reads wait in the queue, and this is measured by the counter 'Avg. Disk Queue Length.' Again, a high number is bad. This counter can be set to average all the disks on a system or you can look at each individual disk. Averaging the disks may be an interesting exercise, but to understand what is happening on your system, you really need to set a counter for each individual disk.

Processors

Finally, you can look at the activity of the processors using the '% Processor Time' counter. This counter, like the disk one, can either be an average or a precise measure of each processor and once more, the higher the number the worse the performance. Assuming a dedicated SQL Server machine, the average CPU usage can be a useful number. However, if you start looking at machines running more than one instance, with each instance assigned to a particular processor (or set or processors), then you'll need to collect data on the individual processors.

SQL Server health

The above counters point to the general health of the server, but what about the SQL Server itself? There are a few counters here that can indicate how well the server is behaving, but keep in mind that these are only broad indicators. Detailed troubleshooting will require more detailed information to support accurate analysis.

Buffer Cache Hit Ratio

First off, the Buffer Cache Hit Ratio, located in the Buffer Manager, will show what percentages of pages were found in memory, thereby avoiding a disk read. In this instance, higher is better. Under most circumstances, with a well configured machine, you should see a buffer cache hit ratio above 90% and probably average over 95%. Lower numbers indicate a lot of disk reads which will slow things down.

Full Scans/Sec

Next, as a general measure of health, it's good to look at the Full Scans/Sec counter in Access Methods. This is basically the number of table or index scans that the system is experiencing. A high number here shows either poorly written stored procedures or bad indexing. Either way, you need to get to work identifying the source of the problem.

Lock Requests/Sec

Under Locks the Lock Requests/Sec counter can show the number of new locks and lock conversions that are taking place on the system. This counter can be general, showing Total Locks, or it can get extremely specific, counting row ID locks (RID) in 2005 or key, file & page locks in both SQL Server 2000 & 2005. While higher numbers may be bad, depending on the circumstances, they may also be an indicator of just a lot of use on the system. If this number is spiking or growing over time, you will need to pursue more details to ascertain whether or not you have a problem.

Deadlock/Sec

Also under Locks, you may want to put the Deadlock/Sec counter on if you're experiencing deadlocks. Deadlocks, by their very nature are indicative of performance problems that require, at the least, procedure tuning and review, and a check on the indexes of a machine. Other steps may be required.

User Connections

Not actually an indicator, but a very useful measure in combination with all the other counters, is the User Connections counter under General Statistics. It's not that this number will necessarily indicate a problem on your server, but it helps in conjunction with the other counters to identify where real problems exist. For example, say you have a larger than normal number of locks. Check the number of user connections. If it's higher than normal, then you're probably just experiencing a spike in usage, but if it's average or below average, then you may have a problem and it's time for more detailed investigation.

Batch Requests/Sec

The last general measure is Batch Requests/Sec under SQL Statistics. This measure quite simply is the number of requests coming in to the server. This is a very general number and may be indicative of nothing more than high use, but it's a good value to track over time because it can show you how your system use is scaling. It can also be used to indicate when you have peaks and valleys in the number of user requests on the system.

The counters outlined above are only the beginning of the metrics you can use to get a general measure of system performance. Other available counters will enable you to drill down into specifics within SQL Server or the server itself. After determining what the OS and the Server are up to, you will need to look inside at what the queries are doing. This is where Profiler comes into play.

Profiler

Profiler can run, similarly to Performance Monitor, either in a GUI mode or in an automated manner with outputs to files or databases. Sitting and watching the GUI window is usually referred to as SQL-TV. That may be a good way to spot-check issues on a database server, or do some ad hoc troubleshooting, but for real performance monitoring you need to set up an automated process and capture the data for processing later. Profiler collects information on events within SQL Server. The broad categories of events are as follows:

- Cursors
- Database
- Errors and Warnings
- Locks
- Objects
- Performance
- Scans
- Security Audit
- Server
- Sessions
- Stored Procedures
- TSQL
- Transactions

Key Events

Each of these categories has a large number of events within it. Rather than detail all the various options, the following is a minimum set of events for capturing basic TSQL performance.

Stored Procedures – RPC:Completed
This records the end point of a remote procedure call (RPC). These are the more common events you'll see when an application is running against your database.

Stored Procedures – SP:Completed
These would be calls against procedures from the system itself, meaning you've logged into SQL Server and you're calling procedures through query analyzer, or the server is calling itself from a SQL Agent process.

TSQL – SQL Batch:Completed
These events are registered by TSQL statements running locally against the server which is not the same as a stored procedure call, for example: SELECT * FROM tablename.

Data Columns

Each of these events can then collect a large number of columns of information, each one may or may not be collected from a given event, depending on the event and column in question and each one may collect different data from the event, again depending on the event and column in question. These columns include but are not limited to:

TextData In the events listed above this column represents the text of the stored procedure call, including the parameters used for the individual call, or the SQL batch statement executed.

ApplicationName This may or may not be filled in, depending on the connections string used by the calling application. In order to facilitate trouble shooting and performance tuning, it's worth making it a standard within your organization to require this as part of the connection from any application.

LoginName The NT domain and user name that executed the procedure or SQL batch

CPU This is an actual measure of CPU time, expressed in milliseconds, used by the event in question.

Reads These are the count of read operations against the logical disk made by the event being captured.

Writes	Unlike the Reads, this is the physical writes performed by the procedure or SQL batch.
Duration	This is the length of time that the event captured took to complete. In SQL Server 2000 this piece of data is in milliseconds. As of SQL Server 2005, this has been changed and is now recorded in microseconds. Keep this in mind if you're comparing the performance of a query between the two servers using Trace events.
SPID	The server process ID of the event captured. This can sometimes be used to trace a chain of events.
StartTime	This records the start time of the event.

In short, a great deal of information can be gleaned from Profiler. You may not be aware that in previous versions of SQL Server, running Trace – as Profiler used to be called– against a system could bring the system to its knees before you gathered enough information to get a meaningful set of data. This is no longer true. It is possible to turn on enough events and columns to impact the performance of a system but with a reasonable configuration Profiler will use much less than 1% of system resources.

That does not mean that you should load up counters on the GUI and sit back to watch your server. This will add some load and can be easily avoided. Instead, take advantage of the extended stored procedures that are available for creating and managing SQL Traces. These will allow you to gather data and write it out to disk (either a local one or a remote one). This means that you'll have the data in a transportable format that you can import into databases or spreadsheets to explore, search and clean to your heart's content. You can also write the results directly to a database, but I've generally found this to be slower, therefore having more impact on the server, than writing to a file. This is supported by recommendations in the BOL. Here is a basic script to create a trace for output to a file:

In order to further limit the data collected, you may want to add filters to restrict by application or login in order to eliminate noise:

```
EXEC sp_trace_setfilter
      @trace_id,
      @columnid,
      @logicaloperator,
      @comparisonoperator,
      @value
```

So, for example to keep any trace events from intruding on the data collection above, we could add:

```
EXEC sp_trace_setfiter
   @trace_id = @TraceId,
   @columnid = 10, --app name column
   @logicaloperator = 1, -- logical "or"
   @comparisonoperator = 0, -- equals
   @value = N'SQL Profiler'
```

The output can be loaded into the SQL Server Profiler GUI for browsing or you can run this procedure to import the trace data into a table:

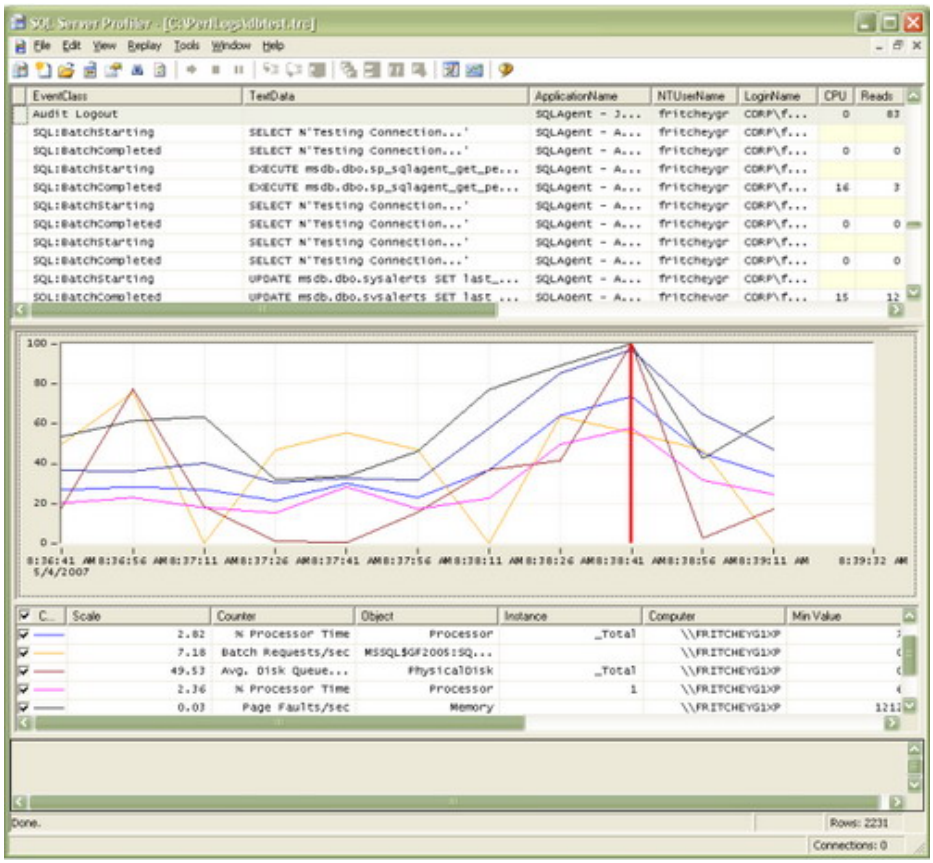
```
SELECT * INTO temp_trc
   FROM fn_trace_gettable('c:\temp\my_trace.trc', DEFAULT);
```

Evaluating Profiler Data

Now that you've collected all this data about your system, what do you do with it? There are many ways in which you can use Profiler data. The simplest approach is to use the data from the Duration column as a measure of how slowly a procedure is performing. After collecting the information and moving it into a table, you can begin writing queries against it. Grouping by stored procedure name and stripping off the parameters from the string will allow you to use aggregates, average, maximum, and minimum, to outline the poorest performing procedures

on the system. From there you can look at their CPU or I/O and query plans in order to identify the tuning opportunities available to you.

With the release of SQL Server 2005, one additional piece of functionality was added to Profiler which will radically increase your ability to identify where you have performance issues. You now have the ability to load a performance monitor log file and a profiler trace file into the Profiler. This allows you to identify points of time when the server was under load and see what was occurring within the database at the same time.



Third Party Tools

There is an entire industry built around monitoring servers and databases. Most of these use the same counters that you can access through Performance monitor or Profiler. A few of these tools find ways to monitor packets or logs or memory and provide a slightly different set of measures. However, most of them do things that you could eventually do for yourself,. What they offer is the ability to set up monitoring and alerts and take advantage of all the work that they've put into recording the data over time, setting up reports, building enterprise level alerts, etc. If you have more than a handful of servers to monitor, doing the research and finding a third party tool that works in a manner that you find acceptable is worth the time and effort it will save you.

Tuning Performance

In short, this is setting or changing the server configuration, the database indexing or storage or the T-SQL code in order to achieve enhanced performance. After gathering information about what is happening on your machine, you can begin the process of identifying where you are experiencing slow performance. Once you've identified the source you can do something about it.

Server Performance

The quickest possible solution here is to get more and faster CPUs running on more and faster memory against bigger and faster disks. There, we're done. What's that, you've got a big honking machine and it isn't performing properly, or you've got a limited budget, so you need to squeeze more from the machine that you have? OK. Listed below are some areas where you can change settings in order to make a difference to how your machine is configured. Most of these are suggestions as each and every set of circumstance is unique.

Rather than let SQL Server manage memory allocation, and grow and shrink it as necessary, simply determine the maximum memory that you can allocate to the server and fix the memory allocation at this point. Variable memory is only helpful if you're running SQL Server in a mixed environment with other applications. (This, by the way, will lead to poor performance as the server has to contend with other applications for precious memory, cpu, and disk resources.)

You should reserve somewhere between 1GB and 2GB of memory for the O/S, depending on the amount of memory on the system.

You should enable AWE (Address Windowing Extensions) on systems with more than 4GB of memory.

Make sure that the maximum number of processors (MAX DOP) is set to the actual number of physical processors on your machine, not the number of hyper-threads.

One of the most consistent bottlenecks in SQL Server is the tempdb. This is used when applications create objects such as temporary tables, but it is also used when rebuilding indexes and sorting data. Create one tempdb data file for each physical processor on the machine. Where possible, isolate the tempdb files to their own drives.

Database Performance

Database performance almost always comes down to I/O. How much data can you get in/out of the system and how quickly? First, and most important, will be your actual database design. It can't be over-emphasized that a poorly constructed database, no matter how sophisticated the server, will perform badly.

Indexing

To start with, you need to plan an indexing strategy at the same time as you plan the database. First, and most important, is the clustered index. As a rule, every table in an online transactional processing (OLTP) system should get a clustered index. There are exceptions, but the exceptions should be exceptional. You can only put one clustered index on a given table, so the proper selection of exactly what column or columns to place it on is extremely important. By default, most tools simply place the clustered index on the primary key. This may well be the appropriate place to put a clustered index, but you need to evaluate how the data is most likely to be accessed.

It makes sense to leave the cluster on the primary key if that primary key provides the natural access path – the most common field used to either search the table or relate the table to another.

Changing the parameters slightly, if the access path is mostly through another column, say Company Name, this may make a more appropriate clustered index. Another situation is when the table is no longer at the top of the chain, but somewhere in the middle and the most likely avenue of selection for the data is going to come through the foreign key to a parent table. Then the foreign key column becomes a good candidate for the clustered index. A further wrinkle could be added by needing to get the related data in this child table, but based on a date. This would result in a clustered index composed of two columns: the foreign key and the date. As you can see, in a few sentences a number of options were laid out. You need to think this through as you design the system.

You may also identify, either during design, testing or monitoring the production system, that other indexes are needed. While multiple indexes on any given table may be needed, you need to keep in mind that each index adds overhead to the system because these values have to be maintained as the data gets modified, which includes inserts, updates and deletes. Further, since indexes are stored by pointing to the clustered index (one of the many reasons you need a clustered index on the table), changes to the clustered index can result in a cascade through all the other indexes on a table. Because of all this, while indexes are good and necessary, restraint must be exercised in their application and use.

Files and FileGroups

Other factors that can affect the performance of your system include the way the files and file groups are laid out. You should be creating multiple files for your databases to optimize performance. A baseline for this would be to create one file for the logs, another for the data (defined by the clustered index), and another for non-clustered indexes. Additional files may be necessary to separate out BLOB data or XML data or unusually active tables, each onto its own file and, where possible, onto its own disk. This has been found to be true even on SAN systems because distributing the load takes further advantage of the architecture of the SAN.

Normalization

The data stored and the data retrieved should be defined appropriately. This means normalizing the storage. If you simply want a flat file, don't put it into a relational database system. You're paying for overhead you don't need and sacrificing benefits that you could achieve. That said, targeted denormalization, picking some fields to duplicate rather than maintaining a perfect third normal form system, will provide some performance benefits.

Data Types

Define the data types that you need, not what you think you might need someday. A phone number is a string, not a number. Define the length of field that you need and enforce that length.

Other Issues

Simple things can make a difference too:

- Turn auto-shrink off.
- Make sure auto-update of statistics is turned on.
- If a database is read-only, set it to read-only.
- Use triggers very judiciously. They mostly operate in the background, making them difficult to monitor and troubleshoot.
- Be very careful of autogrowth settings on the database. 10% autogrowth will be fine when the database is 500mb. It makes a huge difference when the system is 50gb. For larger databases, change the setting to grow by a fixed amount, rather than a percentage of total database size.

TSQL Performance

After you've configured your server and built a functional database, you'll need to move data in and out of it. Most of this will be through TSQL queries. These queries should be defined within stored procedures and will make use of views and user-defined functions to manipulate and access sets of data. The most important part of this concept is the set. Most people think in terms of pieces of data instead of sets of data. This conceptual shift – to manipulating the data in batch instead of row by row – delivers the biggest performance benefits when working with TSQL. Learning TSQL syntax and set-based querying methodologies up front will provide more performance benefits by having well written procedures up front. This is much easier than attempting to tune or fix hundreds or even thousands of poorly written procedures after the fact.

Writing TSQL queries that perform well isn't always as easy as it sounds. The target needs to be to work with the TSQL compiler and optimizer, processes internal to the SQL Server itself, to provide them with queries that they can tune to deliver your data. You must then start with the basics and get the simple stuff right at the start.

- Make sure that your queries are written to manipulate only the data you need.
- Ensure that simple things like qualifying database objects by their owning user or schema are a regular part of your coding practices
- Learn and use the latest ANSI style of syntax employed by SQL Server (ANSI 99 for 2005, ANSI 92 for 2000).
- Avoid cursors as much as possible. While there are some good uses for them, they usually become a crutch used instead of learning how to manipulate the data in sets.

- Remember that transactions and transaction processing within the procedure should be kept as small as practicable. For example, if you need to do an insert and read some data, separate the read from the insert.
- Minimize the dependence in your code on constructs such as table variables or temporary tables. Again, these very useful tools frequently substitute a piece-meal approach for one that emphasizes sets of data.
- When writing stored procedures, things as simple as making the data type of parameters match the data type of the columns being referenced can make a big difference.

Once you've identified a stored procedure or query as being problematic, you'll want to tune it. This is where a query plan comes into play. SQL Server 2000 can display either a graphical plan (estimated and actual) or a text plan. SQL Server 2005 adds on the XML plan. There are advantages to each type of plan. The graphical plan can be a quick and easy way to peruse the actions a query has put the system through. Especially useful is the ability to display the estimated query plan, which could identify problems without having to actually execute a query. Text and XML query plans present more data immediately to the user and can be searched or parsed through code, allowing for some automation of tuning if so desired. Some of the things to look for in your query plans are scans, work tables, or hash joins. Each of these can usually be fixed by adding an index to the system or adjusting the query so that the join or where clause takes advantage of an existing index.

Client Access

All the good work done within SQL Server can be undone by a poorly written piece of client code. More often than not, the DBA can't write that code for the company. However, you can monitor, mostly through Profiler, what that code is doing to your system and how it is doing it. This is where you can make a difference.

If transactions are managed by code instead of by the database, observe these transactions to ensure that they are as short as possible and that any data access is isolated from other client side processing.

A classic error is to open a database transaction and then wait for user input. Be sure that the application is using the procedures you provided in the manner in which they were meant. You may have a large query that frequently returns a lot of data. Talking to the developers you may find that they only need one or two columns from this result set. Providing them with a modified or new procedure can save lots of processing time.

Be sure that the latest ADO or ADO.NET is in use. Be sure that the connections being made are using the settings you expect, since things such as the connection timeout can be different from the database timeout, resulting in odd behavior within your application. Again, moving only the data you need, and only when you need it, should be a working maxim.

Testing Performance

Instead of discovering problems only when the application hits production, try loading up the database with some data and run tests against it. While it is time consuming, database testing is as easy as that. First, you need a good understanding of the data. Just how many rows of what type of data are you likely to see? Then you need a good understanding of your data model so that you know that 'x' parents have so many children on various tables of type 'y'. This isn't a technical issue. It's primarily a series of questions put to your business.

Once this is defined, you either have to define your transactions, or put Profiler to work capturing a set of transactions from the application. Obviously the second approach is preferable since the data will be much more accurate.

For this you can use Profiler to replay the transactions you captured against a system with a lot more data in it. This will allow you to identify where you'll be experiencing performance problems arising from the amount of data in the system, your indexes, table design and your TSQL code. What this won't show you is what happens when you have multiple users hitting the system at the same time.

This is a much more difficult type of testing. I'd recommend looking into obtaining a third party tool such as Quest Benchmark Factory or Idera Diagnostic Manager. These allow you to take the series of transactions you've recorded and run them through, with multiple users simultaneously hitting the system. Testing of this type allows

you to identify performance problems due to contention, deadlocks, and again your TSQL code, table design and indexes.

If nothing else, open up each query and look at the query plan to identify areas that might cause you trouble down the road. While that table scan is hitting 75 rows, it doesn't really affect performance, but when it's hitting 75,000 or 75 million rows, you'll probably notice. While running through all these tests, use Performance Monitor and Profiler to capture the data. This can then be used over time to track how well your performance tuning is working out.

Suggested Reading

All of this only scratches the surface of what may be required to accurately assess and rectify performance issues in SQL Server. For more information consult books such as:

- "Inside SQL Server 2000" by Kalen Delaney
- "Inside SQL Server 2005: The Storage Engine" by Kalen Delaney
- "Inside SQL Server 2005: TSQL Querying" by Itzik Ben-Gan
- "Inside SQL Server 2005: TSQL Programming" by Itzik Ben-Gan

There are also fantastic web sites, in addition to Simple-Talk, like:

- Troubleshooting Performance Problems in SQL Server 2005
- SQL Server Performance
- SQL Server 2005 Waits and Queues
- SQL Server Central, (specifically: Performance Monitoring tutorial Part1)

Chapter 8: SQL Server Replication Crib Sheet

Replication is intended to be a way of distributing data automatically from a source database to one or more recipient databases. As such, it can have obvious uses in a distributed system. It has also been used to implement high-availability systems. It is not useful for one-off synchronization, or for simply copying data. It is intended as a long-term data relationship between databases. Typical uses are in:

- Data warehousing and reporting
- Integrating data from several, possibly only partially connected, sites
- Improving scalability and availability
- Integrating heterogeneous data from other databases via OLE DB, integrating data from mobile users, getting data to and from Point-Of-Sale systems
- Offloading/delegating batch processing tasks.

Examples of the use of replication within an application could be:

- Distributing data 'owned' by a particular application to other applications that are 'consumers' of that data. (For example, sales records to reporting services, manufacturing stock levels to purchasing systems.)
- Creating several instances of a database to distribute load on it
- Updating a central database with information from a number of remote Laptops that might be only partially connected, and to resynchronise the laptops.

There are three methods of replication: Snapshot, Transactional, and Merge. These are provided to try to meet the wide range of business requirements for replication.

Replication uses a 'Magazine Publishing' vocabulary. Anything from an 'Article' to an entire database can be replicated. An Article is the smallest component of distribution and can be a table, procedure or function. If it is a table, then a filter can be applied to it so that only certain rows or columns are replicated. This 'Magazine Publishing' analogy can be confusing because, in replication, a Subscriber can sometimes make updates, and a Publisher usually sends out incremental changes to the articles in a publication.

- A **'Publisher'** maintains the original copy of the data. It holds the definition of the 'Publication', which defines the 'articles' that are to be 'published'. (The database with the original location of the data determines what is to be distributed).
- A '**Subscriber**' receives the articles from a publisher. It can subscribe to one or more publications. Any database can take on either role or even both roles at once.
- A **Distributor** is a specialist database that runs the 'Replication agents'.

Replication is not part of the SQL Server engine, but an external application. This makes it much easier to involve other database systems in replication. Any SQL Server database, or other database system with an OLE DB provider, can be a publisher, or subscriber in snapshot, or transactional replication.

It is essential to plan out the replication in detail as a first stage, and to be very certain of the type of replication you wish to implement. A common mistake is to use replication in cases where a much less complex solution is possible.

A problem with production systems using replication is the difficulty of restoring the topology after a disaster. This requires a fully documented recovery strategy, which has to be periodically tested and practiced. This means that the whole replication topology and configuration must be scripted so it can be re-created in an emergency, even if the system was originally built using the GUI tools. The Object Browser (or Enterprise Manager) makes the initial deployment relatively simple and there are plenty of step-by-step guides, but it is not the best option when trying to restore an existing topology, and settings, in order to achieve a recovery from major failures.

Problems often follow from developers adding or dropping articles, changing publication properties, and changing schema on published databases. It is therefore best to create the replication once the design of the publisher is relatively stable.

Replication topologies

In most topologies, it makes sense for publishers, distributors, and subscribers to be on separate physical hardware.

Central Publisher

The commonest form of replication is to have a single publisher with the source data, with one or more subscribers. The Distributor database can be on the same server, but preferably a different one.

Central Subscriber

Often, where the data from several databases need to be 'warehoused' centrally in an OLAP or reporting database, one will find a single 'reporting' database subscribing to several publications.

One can come across other topologies such as 'bi-directional' and 'peer to peer' which are really special cases of Central Publisher or Central Subscriber and use transactional replication

Publishing Subscriber

The distribution of data can be relayed to other subscribers via a publishing subscriber. This allows replication to be implemented over low-bandwidth WANs to a subscriber that, in turn, distributes it to other servers within its high-bandwidth LAN.

Replication Methods

Replication begins with the initial synchronization of the published objects between the Publisher and Subscribers, using a snapshot. A snapshot is a copy of all of the objects and data specified by a publication. After the snapshot is created on the publisher, it is delivered to the Subscribers via the distributor.

For Snapshot replication, this is sufficient. For other types of replication, all subsequent data changes to the publication flow to the Subscriber as they happen, in a queue, or on request.

Snapshot Replication

The snapshot replication process provides the initial synchronization for transactional and merge publications. However, in several cases this initial synchronization is all that is necessary. This would include circumstances where data hardly changes, or if the latest version of the data is not essential to the subscriber, where the amount of data is small, or if a large number of changes takes place rapidly.

Snapshot replication involves copying the articles that make up the publication. Normally, if they exist already on the subscriber, they are over-written, though this behavior can be changed. Snapshot replication is more expensive in terms of overhead and network traffic and only takes place at intervals. Because locks are held during snapshot replication, this can impact other users of the subscriber database. It is therefore more suitable for static data and enumerations. In SQL Server 2005, several articles can be processed in parallel, and interrupted snapshots can be recommenced from the point of interruption. Snapshots can be queued or immediate.

Data changes are not tracked for snapshot replication; each time a snapshot is applied, it completely overwrites the existing data.

Transactional Replication

Transactional Replication is used if:

- Changes to the data must be propagated immediately
- The database application taking out a subscription needs to react to every change
- The Publisher has a very high volume of insert, update, and delete activity
- The Publisher or Subscriber is a different database application reached via OLE DB.

Essentially, Transaction Replication distributes data in one direction, but it does offer options that allow updates at the Subscriber. Once a snapshot replication has synchronized the subscribers with the publisher, all committed transactions on the publisher are then propagated to the subscribers in sequence, via distributed transactions. One can select a queued update or immediate, depending on requirements.

Peer-to-peer Replication

This is a special type of transactional replication in which every participant is both a publisher and subscriber (2005 Enterprise only) and is most useful for up to ten databases in a load-balancing or high-availability group.

Bidirectional Replication

Bidirectional Replication occurs when two databases replicate the same articles to each other via a distributor. There must be loopback detection. Data conflicts aren't handled and the replication must be implemented in code, since the GUI doesn't support it.

Transactional Replication tracks changes through the SQL Server transaction log

Merge Replication

Merge Replication allows various sites to work autonomously and later merge updates into a single, uniform result.

Merge Replication is complex, but provides the means to implement part of a high-availability system, as well as its original purpose of serving mobile and disconnected users. It is designed for cases where the publishers are not in constant communication with the subscribers. After the initial snapshot synchronization, subsequent changes are tracked locally with triggers, and the databases are merged when in contact, using a series of rules to resolve all possible conflicts.

Merge Replication is used when several Subscribers might need to update the same data at various times and propagate those changes back to the Publisher and thence to other Subscribers. It is also required in applications that involve Subscribers receiving data, making changes offline, and finally reconnecting with the publisher to synchronize changes with the Publisher and other Subscribers.

To make this possible, each Subscriber requires a different partition of data and there has to be a set of rules to determine how every conflict that takes place in the update of the data is detected and resolved. These conflicts occur when the data is merged because there can be no 'locking' and so the same data may have been updated by the Publisher and by more than one Subscriber.

Merge Replication does not use transactions. Merge Replication uses a set of conflict-resolution rules to deal with all the problems that occur when two databases alter the same data in different ways, before updating the subscribers with a 'consensus' version. It normally works on a row-by-row basis but can group rows of related information into a logical record. One can specify the order in which 'articles' are processed during synchronization.

Merge Replication tracks changes through triggers and metadata tables.

Replication Agents

Replication is done by several different agents – separate applications, each responsible for part of the process. The replication agents should not be run under the SQL Server Agent account in a production system. Instead, they need the minimal permissions necessary to perform their function.

SQL Server Agent

This manages the overall replication process via SQL Server Agent jobs.

The Snapshot agent

Snapshot.exe executes on the Distributor. It extracts the schema and data defined by the publication, which is then sent to the subscriber via a 'snapshot folder'. It also updates status information on the distribution database. It is used in all forms of replication

The Log Reader Agent

LogRead.exe is used in transactional replication to extract relevant committed transactions from the publisher's log, repackage them and send them to the distributor in the correct sequence.

Distribution Agent

Distrib.exe takes the snapshots and log entries from the agents we've described, and dispatches them to the subscribers.

Merge Agent

ReplMer.exe is used only in Merge Replication to send a snapshot when the subscriber is initialized, and also exchanges transactions between publisher and subscriber

Queue Reader Agent

QrDrSvc.exe is used to queue the updates in transactional or snapshot replication when queuing has been specified.

Monitoring Replication

Many problems associated with replication can be avoided by regular checks. The most obvious check is to make sure that the data has been transferred as expected. Periodic checks with SQL Compare and SQL Data Compare can be very useful, in addition to the tools that come with Replication. Additionally, the replication processes and jobs need to be checked to make sure they are working.

Checking throughput

The performance of replication must be regularly monitored and performance-tuned as necessary.

The Replication Monitor is used to check on the operational state of publications and inspect the history and errors. Right-clicking the replication node in Object Explorer will gain access to it.

One of the most important concerns is the time delay, or latency, of transactions from the publications appearing in the subscriber database. At times of high transaction throughput on the publisher database, bottlenecks can

occur. Whereas the stored procedure **sp_browseReplCmds** on the distribution database can tell you how far behind the synchronization is at any particular time, one cannot determine where the problems lies, solely from the data. Tracer tokens are now used to measure the actual throughput of the replication architecture at any particular time to help diagnose such bottlenecks.

Validating

There is always an element of doubt as to whether the replication has entirely worked. There are stored procedures provided to compare the 'articles' on the publisher and subscribers to make sure they are the same.

The **sp_publication_validation** stored procedure validates the data associated with each article by calling **sp_article_validation** (after the articles associated with a publication have been activated). The **sp_article_validation** stored procedure invokes **sp_table_validation** stored procedure, which calculates the number of lines and, optionally, the checksum of the published table. It is considered good practice to perform a daily row-count and weekly checksum. SQL Data Compare is ideal for mending a broken replication.

The Distribution Agent raises the '20574' system message if validation fails, or the '20575' system message if it passes. The Distribution Agent will replicate changes to a subscriber even if the validation shows that the subscriber is out of synchronization. It is a good policy to configure the Replication Alert on the '20574' message so as to send E-Mail, Pager, or Network notification.

This validation approach will only work within certain restrictions. For example, it will not work if certain filters have been applied. They should be used with caution.

Changing the settings

It is best to use the default replication settings unless there are clear performance gains to be made, or if the application design forces the issue. However, one cannot assume that the changes will be generally beneficial to the entire topology without comprehensive testing.

Articles

Articles are the smallest unit of a publication. An article can be a table, view, stored Procedure or function. Where an article is based on a table or view, it can contain all the data or just part of it. These filters of two types.: More common are the static 'WHERE' clauses, but filters can be used dynamically in Merge Replication to publish different 'content' (rows) to different 'subscribers' (databases receiving data). These latter Filters are called 'Dynamic' and can be simple Row Filters, or Join Filters, where the selection of rows to publish is based on a join with other tables, rather than a simple WHERE clause.

Normally, any alteration to an article that is a table is propagated to all the subscribers. You can also opt to propagate schema objects associated with the article such as indexes, constraints, triggers, collation and extended properties.

Updating articles

In Merge replication, the subscriber can update the article. This is of course a recipe for conflict and these have to be resolved automatically. When the Merge Agent comes across a row that might have changed recently, it examines the history or 'lineage' of each site's version of the row, to see if there is a conflict. If so, then the update that is finally used has to be based on either:

- A 'first wins' resolution,
- a user-specified priority scheme to determine the update to select,
- A customized resolution, using COM and stored procedures.

The 'lineage' is a history of changes in a table row in **MSmerge_contents**, which is maintained automatically when a user updates a row. Each column contains one entry for each site that has updated the row.

Conflicts to the data in the base table can occur within a column or a row. Most usual are column-tracked articles. This means that, within any row, updates are only recognized as conflicts if the same column is updated by more than one subscriber. Occasionally, however, the business rules of the application may treat simultaneous changes to the any column within the row as a conflict – in which case row-level tracking is used.

Programming Replication Topologies

Replication agents and replication topologies can be administered and monitored remotely via SQL Scripts or RMO scripts. The task of building and maintaining replication topologies is made much easier if all parts of the deployments are scripted, even if this is done after the other methods such as wizards or RMO are used for the initial operation.

There are other uses for a replication script. It will be required in end-user applications where, for example, such as a pull subscription is synchronized when the user clicks a button, or where a routine administration task such as monitoring replication throughput is performed from a custom console. It is also generally used for writing customized business rules that are executed when a merge subscription is synchronized.

One can use the Object Explorer in SSMS, or the Enterprise Manager in earlier versions of SQL Server, to set up replication. BOL provide worked examples. Alternatively, RMO can be used with VB or C# to script the replication. Whichever system you use, it is a good idea to use the Transact SQL script as the reference for the replication topology you create

Ultimately, the functionality in a replication topology is provided by system-stored procedures. The easiest approach is to use Transact-SQL script files to perform a logical sequence of replication tasks, because it provides a permanent, repeatable copy of the steps used to deploy the replication topology that can, for example, be used to configure more than one subscriber. It also provides a measure of documentation and disaster-recovery. A script can be stored as a query object in a SQL Server Management Studio project.

Replication scripts can be created either by hand, by the script generation of the replication wizards in SQL Server Management Studio, or by using Replication Management Objects (RMOs) to programmatically generate the script to create an RMO object.

When creating scripts to configure replication, it is best to use Windows Authentication This avoids storing security credentials in the script file. Otherwise you must secure the script file.

Further reading:

- SQL Server Replication
- For the details of implementing replication, all the steps are documented here in Implementing Replication
- details on configuring and maintaining a replication are here Configuring and Maintaining replication
- For details on peer-to-peer replication, read Peer-to-Peer Transactional Replication
- Some of the wider issues and dangers of replication are discussed here Data Replication as an Enterprise SOA Antipattern, in the excellent Microsoft Architecture Journal

Chapter 9: Entity Framework Crib Sheet

Database developers and application developers each need to model business processes and data differently. Database developers use a relational model to represent data and develop stored procedures to represent processes. Application developers use the richness of object-oriented models to work with business data and business processes. This mismatch between the relational world and the object-oriented world can be addressed by using the ADO.NET Entity Framework. This article is based on the ADO.NET Entity Framework that ships with Visual Studio.NET 2008 Service Pack 1 Beta.

The ADO.NET entity framework is an evolution of ADO.NET designed to provide object relational mapping between conceptual entities and the data store. Conceptual entities are standard .NET classes. The ADO.NET entity framework provides a mapping technology to map these conceptual entities to the data store schema of your database. It allows an application developer to work with the conceptual entities without having to be concerned about the underlying data model.

ADO.NET Entity Data Model

The ADO.NET entity framework uses an Entity Data Model. The Entity Data Model consists of the storage schema, the conceptual schema, the mapping schema and the entity classes.

Storage Schema Definition (SSDL)

The storage schema describes, in an XML definition, the schema of the data store. This is the exact schema of your data store. The storage schema definition can be generated directly from the data store. The Storage schema describes the schema of a database table as an entity type. The following XML fragment shows the definition for a Customer table:

```
<EntityType Name="Customer">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar" Nullable="false"
                                        MaxLength="25" />
  <Property Name="City" Type="nvarchar" MaxLength="25" />

  <Property Name="Country" Type="nvarchar" Nullable="false"
                                        MaxLength="25" />
  <Property Name="IsCorporate" Type="bit" Nullable="false" />
  <Property Name="CompanyName" Type="nvarchar" MaxLength="25" />
  <Property Name="Designation" Type="nvarchar" MaxLength="25" />
  <Property Name="LastModified" Type="timestamp"
                            StoreGeneratedPattern="Computed" />
</EntityType>
```

The storage schema also describes relationships between tables as an association. The following definition for an association shows that the Customer table has a relationship with an Order table:

```
<Association Name="FK_Orders_Customers">
  <End Role="Customer" Type="ContosoModel.Store.Customer" Multiplicity="1" />
  <End Role="Order" Type="ContosoModel.Store.Order" Multiplicity="*" />
<ReferentialConstraint>
    <Principal Role="Customer">
      <PropertyRef Name="ID" />
    </Principal>
    <Dependent Role="Order">
```

```
      <PropertyRef Name="CustomerID" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

The storage schema also refers to any stored procedures in the data store. The following XML definition defines an existing stored procedure named **SelectByCountry** that accepts a single parameter named **ctry:**

```
<Function Name="SelectByCountry" Aggregate="false" BuiltIn="false"
NiladicFunction="false" IsComposable="false"
ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
  <Parameter Name="ctry" Type="varchar" Mode="In" />
</Function>
```

Finally, all the defined entity types and associations are mapped into an entity container. The entity types are grouped into an entity set and the associations are grouped into an association set.

```
<EntityContainer Name="dbo">
  <EntitySet Name="Customer" EntityType="ContosoModel.Store.Customer"
store:Type="Tables" />
  <EntitySet Name="Order" EntityType="ContosoModel.Store.Order"
store:Type="Tables" />
  <AssociationSet Name="FK_Orders_Customers"
Association="ContosoModel.Store.FK_Orders_Customers">
    <End Role="Customer" EntitySet="Customer" />
    <End Role="Order" EntitySet="Order" />
  </AssociationSet>
</EntityContainer>
```

Conceptual Schema Definition (CSDL)

The conceptual schema describes the conceptual entities as entity types and is defined in XML. This is created to describe the schema of the conceptual entities. Conceptual entity types are .NET classes. Though this schema can be auto-generated, mostly the application developer would alter the conceptual schema to reflect the model used for the conceptual entities. If the conceptual schema is auto-generated it would simply map a conceptual entity type to a table. The following shows the schema for two conceptual entities named Customer and **CorporateCustomer**. The **CorporateCustomer** entity type inherits from the Customer entity type.

```
<EntityType Name="Customer">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Int32" Nullable="false" />
  <Property Name="Name" Type="String" Nullable="false" MaxLength="25"
Unicode="true" FixedLength="false" />
  <Property Name="City" Type="String" MaxLength="25" Unicode="true"
FixedLength="false" ConcurrencyMode="None" />
  <Property Name="Country" Type="String" Nullable="false" MaxLength="25"
Unicode="true" FixedLength="false" />
  <Property Name="LastModified" Type="Binary" MaxLength="8" FixedLength="true"
ConcurrencyMode="None" />
  <NavigationProperty Name="Order"
Relationship="ContosoModel.FK_Orders_Customers" FromRole="Customer"
ToRole="Order" />
</EntityType>

<EntityType Name="CorporateCustomer" BaseType="ContosoModel.Customer" >
  <Property Name="Designation" Type="String" Nullable="true" />
  <Property Name="CompanyName" Type="String" Nullable="true" />
</EntityType>
```

The conceptual schema also describes relationships between entity types as associations. These relationships are references in the conceptual schema for the entity type when an entity contains instances of another entity. For

example, the navigation property in the Customer entity type in the schema above references entities by using an association. The following schema shows the association between the Customer entity type and the Order entity type:

```
<Association Name="FK_Orders_Customers">
  <End Role="Customer" Type="ContosoModel.Customer" Multiplicity="1" />
  <End Role="Order" Type="ContosoModel.Order" Multiplicity="*" />
</Association>
```

Finally, all the defined conceptual entity types are grouped into entity sets, and associations are grouped into association sets. An entity container is defined to include the entity sets, association sets and any function definitions that can be mapped to stored procedures:

```
<EntityContainer Name="ContosoEntities">
  <EntitySet Name="Customer" EntityType="ContosoModel.Customer" />
  <EntitySet Name="Order" EntityType="ContosoModel.Order" />
  <AssociationSet Name="FK_Orders_Customers"
Association="ContosoModel.FK_Orders_Customers">
    <End Role="Customer" EntitySet="Customer" />
    <End Role="Order" EntitySet="Order" />
  </AssociationSet>
  <FunctionImport Name="SelectByCountry" EntitySet="Customer"
ReturnType="Collection(Self.Customer)">
    <Parameter Name="ctry" Mode="In" Type="String" />
  </FunctionImport>
</EntityContainer>
```

Mapping Schema (MSL)

The mapping schema definition is the glue that binds the conceptual model and the data store model. This XML definition contains information on how the conceptual entities, functions and associations are mapped to the storage schema. The following shows the mapping definition to map the Customer and **CorporateCustomer** conceptual entities defined in the conceptual schema to the database table named **Customer** defined in the storage schema:

```
<EntitySetMapping Name="Customer">
  <EntityTypeMapping TypeName="ContosoModel.Customer">
    <MappingFragment StoreEntitySet="Customer">
      <ScalarProperty Name="ID" ColumnName="ID" />
      <ScalarProperty Name="Name" ColumnName="Name" />
      <ScalarProperty Name="City" ColumnName="City" />
      <ScalarProperty Name="Country" ColumnName="Country" />
      <ScalarProperty Name="LastModified" ColumnName="LastModified" />
      <Condition ColumnName="IsCorporate" Value="0" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(ContosoModel.CorporateCustomer)">
    <MappingFragment StoreEntitySet="Customer">
      <ScalarProperty Name="ID" ColumnName="ID" />
      <ScalarProperty Name="LastModified" ColumnName="LastModified" />
      <ScalarProperty Name="CompanyName" ColumnName="CompanyName" />
      <ScalarProperty Name="Designation" ColumnName="Designation" />
      <ScalarProperty Name="Country" ColumnName="Country" />
      <ScalarProperty Name="City" ColumnName="City" />
      <ScalarProperty Name="Name" ColumnName="Name" />
      <Condition ColumnName="IsCorporate" Value="1" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

The associations we define in the conceptual schema are mapped to the associations that we define in the storage schema:

```
<AssociationSetMapping Name="FK_Orders_Customers"
TypeName="ContosoModel.FK_Orders_Customers" StoreEntitySet="Order">
  <EndProperty Name="Customer">
    <ScalarProperty Name="ID" ColumnName="CustomerID" />
  </EndProperty>
  <EndProperty Name="Order">
    <ScalarProperty Name="ID" ColumnName="ID" />
  </EndProperty>
</AssociationSetMapping>
```

The functions we defined in the conceptual schema are mapped to the stored procedures defined in the storage schema:

```
<FunctionImportMapping FunctionImportName="SelectByCountry"
FunctionName="ContosoModel.Store.SelectByCountry" />
```

Entity Classes

Entity classes are .NET classes that are generated from the conceptual schema XML definition. The Entity classes are similar to business entities and will be used by the other layers in your application. When working with the Entity Data Model, any changes to the conceptual schema update the auto generated .NET class definitions. Since these entity classes are partial classes, you can extend the functionality by creating your own business logic for the entity classes. The generated entity classes also contain partial methods to plug in your own validation logic.

The following code shows the auto-generated class for the **CorporateCustomer** entity.

C# Code:

```
[global::System.Data.Objects.DataClasses.EdmEntityTypeAttribute(NamespaceName =
"ContosoModel", Name="CorporateCustomer")]
[global::System.Runtime.Serialization.DataContractAttribute(IsReference=true)]
[global::System.Serializable()]
public partial class CorporateCustomer : Customer
{
    public static CorporateCustomer CreateCorporateCustomer(int ID, string name,
string country)
    {
        CorporateCustomer CorporateCustomer = new CorporateCustomer();
        CorporateCustomer.ID = ID;
        CorporateCustomer.Name = name;
        CorporateCustomer.Country = country;
        return CorporateCustomer;
    }
 [global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute()]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public string Designation
    {
        get
        {
            return this._Designation;
        }
        set
        {
            this.OnDesignationChanging(value);
            this.ReportPropertyChanging("Designation");
            this._Designation =
global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value,
true);
            this.ReportPropertyChanged("Designation");
            this.OnDesignationChanged();
        }
    }
    private string _Designation;
```

```
    partial void OnDesignationChanging(string value);
    partial void OnDesignationChanged();
    //Other property definitions omitted
}
```

You can plug in your own validation logic by providing implementation for the partial methods. For each property in the entity class there will be partial methods that are called, both before a property changes and after it is changed. For example, on the above code let us assume we want to execute validation logic to make some validation checks for a designation set to 'Manager'. Then we can create a partial class definition for the above class and provide implementation for the **OnDesignationChanging** partial method as follows:

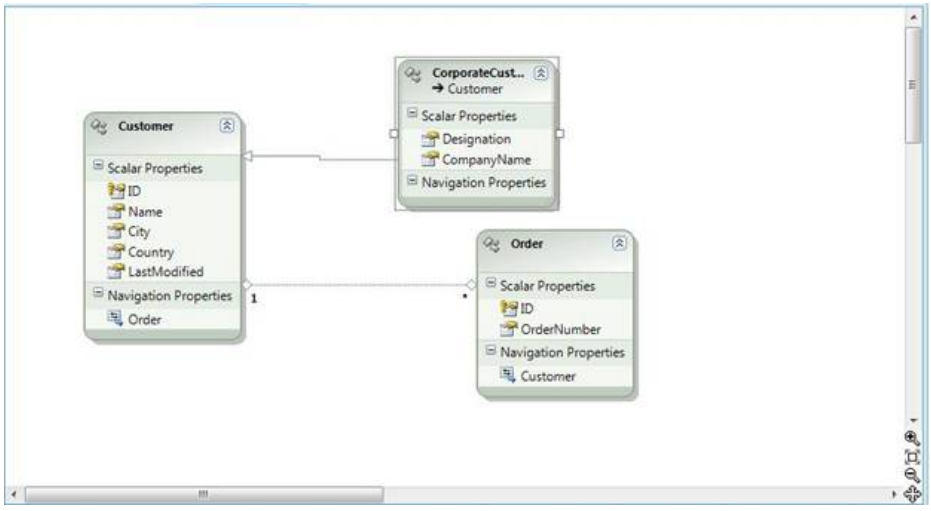
C# Code:

```
partial class CorporateCustomer
{
    partial void OnDesignationChanging(string value)
    {
        if (value.Equals("Manager"))
        {
            //validation checks
        }
    }
}
```

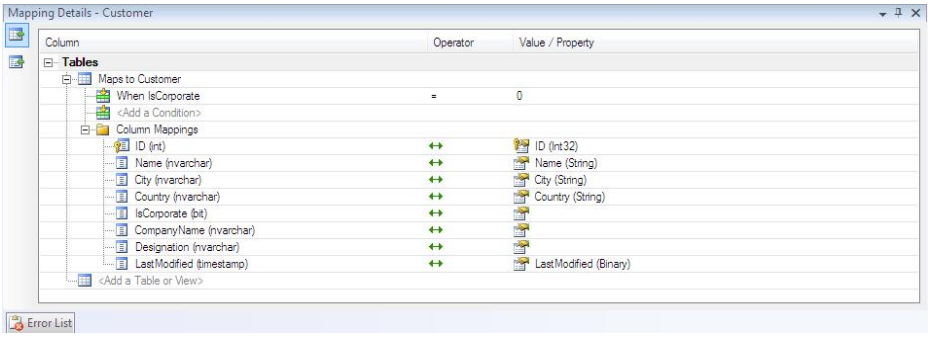
Working with the Designer and Tools

Visual Studio.NET 2008

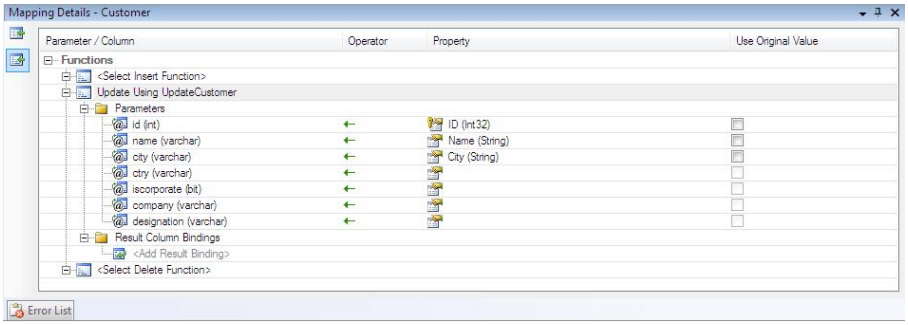
Visual Studio has a project item template for an Entity Data Model. Adding an Entity Data Model to your project allows you to design the conceptual entities and map them to the data store using the Entity Framework designer. You can also create new conceptual entities, create association, and inherit from existing conceptual entities in the designer.



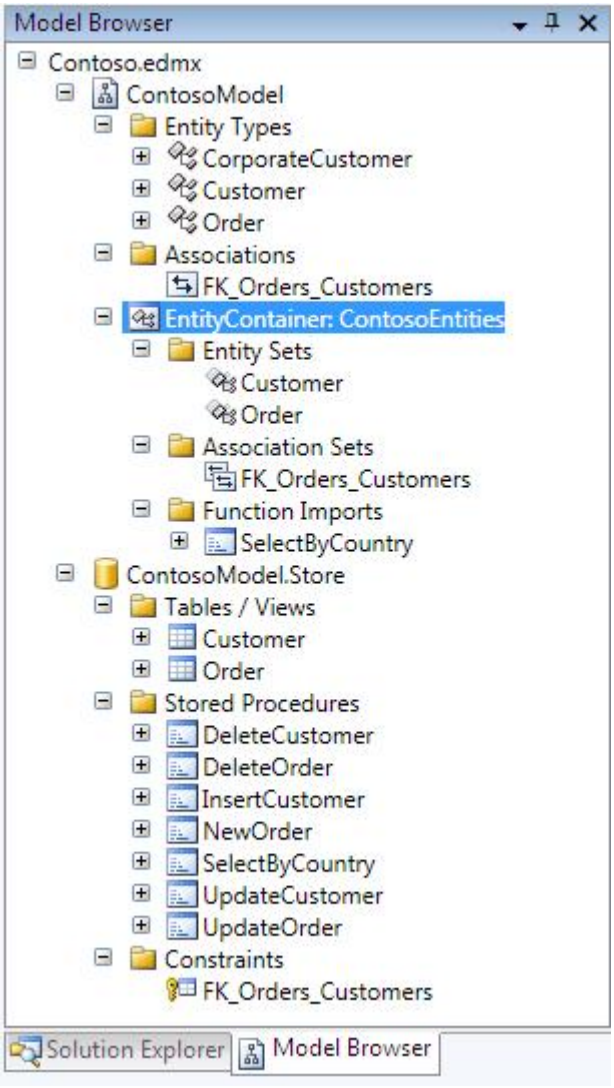
When you work with conceptual entities in the Entity Framework Designer, you can use the Mapping Details window to map the conceptual entity to a data store.



The Mapping Details window also has an alternate view to map your own user defined stored procedures to perform Inserts, Updates and Deletes for your conceptual entity



Visual Studio also has a Model Browser window that shows the conceptual entity types, entity sets and data store. This is similar to the Class Browser window and provides a better view to browse through the Entity Data model.



EdmGen tool

The EdmGen is a command line tool available with Visual Studio.NET 2008. You can use this to generate ADO.NET entity data model schema files and .NET classes for your application. For example, the following command generates the XML files for the conceptual schema, storage schema, mapping schema and entity classes by using the database Contoso:

EdmGen /mode:FullGeneration /project:Contoso /provider:System.Data.SqlClient /connectionstring:"server=.\sqlexpress;integrated security=true;database=Contoso"

You can change the mode for EdmGen to specify different starting points. For example, if you have already created the storage schema, mapping schema and conceptual schema, and you want to generate the entity classes, you can change the mode to "EntityClassGeneration". EdmGen is very useful when you want to configure it to be used together with an automated build process.

Working with data

Using Object Services

Object Services allows you to work with data as conceptual entities (.NET objects). This also allows you to perform CRUD operations on objects without writing query statements that are provider-dependent or targeting

a particular database product. The CRUD operations for entities are executed through the ObjectContext class. The ObjectContext class also manages state for retrieved entities.

Querying data

You have different ways of querying data through Object Services. When data is queried through Object Services, the results are returned as strongly typed objects. This allows you to work with data as working with any strongly typed object or collection in .NET. The simplest way to query data is by using the ObjectQuery type in .NET. The ObjectQuery type represents a query that can be executed against the conceptual model.

Using Query Builder

An ObjectQuery can be constructed by using the available extension methods such as Where, OrderBy, Select etc. The ObjectQuery always work with an ObjectContext (in this instance ContosoEntities). For example the following code block calls the Where extension method to build the query to return only entities from the Customers entity set where the Country property is Australia.

C# Code

```
using (ContosoEntities ent = new ContosoEntities())
{
    ObjectQuery<Customer> query = ent.Customers.Where("it.Country=@ctry");
    query.Parameters.Add(new ObjectParameter("ctry", "Australia"));
    List<Customer> res = query.ToList();
}
```

Using Linq to Entities

The entities in ADO.NET are Linq enabled. You can use the language integrated query (Linq) features available in .NET 3.5 to query data. The following code shows the same query we wrote earlier using Linq to entities.

C# Code

```
using (ContosoEntities ent = new ContosoEntities())
{
    string country = "Australia";
    List<Customer> res = (from c in ent.Customers where c.Country == country
select c).ToList();
}
```

We can also retrieve only a few properties of the Customer entity by using anonymous types and the type inference feature in C# 3.0. For example, the following query retrieves only the Name and the Country of **CorporateCustomer** instances:

```
using (ContosoEntities ent = new ContosoEntities())
{
    var res = (from c in ent.Customer where c is CorporateCustomer select new {
c.Name, c.Country }).ToList();
}
```

The above code will select Customer entity instances that are of the type **CorporateCustomer**. The 'res' variable will hold a reference to a list of anonymous type instances that has the Name and Country properties. You can also use the query builder methods to retrieve a few properties of entity instances.

Using Entity SQL

Entity SQL is a SQL like query language that can be used to query data. Unlike SQL, Entity SQL uses entities, property names and associations defined in the conceptual schema in retrieving data. The following code block shows the query that we wrote using query builder and Linq to entities written using Entity SQL.

C# Code

```
using (ContosoEntities ent = new ContosoEntities())
{
    ObjectQuery<Customer> query = new ObjectQuery<Customer>("SELECT VALUE c FROM
ContosoEntities.Customer as c WHERE c.Country=@ctry", ent);
    query.Parameters.Add(new ObjectParameter("ctry", "Australia"));
    List<Customer> res = query.ToList();
}
```

You can retrieve few properties of your entity instances similar to those we retrieved using Linq to entities. But when using Entity SQL the results can be returned as instances of the type DbDatRecord.

C# Code:

```
using (ContosoEntities ent = new ContosoEntities())
{
    ObjectQuery<DbDataRecord> query = new ObjectQuery<DbDataRecord>("SELECT
c.Name,c.City FROM ContosoEntities.Customer as c WHERE c.Country=@ctry", ent);
    query.Parameters.Add(new ObjectParameter("ctry", "Australia"));
    foreach (DbDataRecord item in query)
    {
        string name = item.GetString(0);
        string city = item.GetString(1);
        //process data
    }
}
```

Using Stored Procedures

If you do not want to rely on the dynamic SQL statements generated by the Entity Framework, you can create your own stored procedures and map them as functions. You can use these functions to query data. For example, in the mapping files we had a stored procedure named SelectByCountry mapped to a function with the same name. We can directly call the function from our code to query data:

C# Code

```
using (ContosoEntities ent = new ContosoEntities())
{
    List<Customer> res = ent.SelectByCountry("Australia").ToList();
}
```

Updating data

Updating entities retrieved is effected by the ObjectContext. The ObjectContext manages state changes on the entity set (inserts, updates and deletes). When the entity set needs to be updated with the data store, you can call the SaveChanges method of the ObjectContext class. This method would execute the required insert, update and delete statements specific to the data store, based on whether the entity instances were newly created, or updated, or deleted.

C# Code:

```
using (ContosoEntities ent = new ContosoEntities())
{
    Customer c1 = ent.Customer.First(c => c.ID == 1);
    c1.Name = "Altered";
    Customer c2 = ent.Customer.First(c => c.ID == 2);
    ent.DeleteObject(c2);
    Customer c3 = new Customer();
    c3.Name = "Customer 3";
    //Set other properties for c3
    ent.AddToCustomer(c3);
    ent.SaveChanges();
}
```

The above code will update the Customer entity instance c1, delete c2 and insert the instance c3 as a new entity to the data store. All the inserts, updates and deletes would occur when the SaveChanges method is invoked.

While updating data, the Entity Framework uses optimistic concurrency. Rows are not locked when entities are retrieved, but at the time of updating it checks whether data was changed. You can control the properties of the entity that will be part of the concurrency check by setting the Concurrency Mode to Fixed. If the Concurrency Mode for a property of an entity is set to fixed, then the auto-generated SQL statement would compare the original value that was retrieved with the current value in the data store. If there are any conflicts, an exception would be thrown.

If you do not wish to use the auto-generated SQL statements for persisting changes, you can define your own stored procedures for inserting updating and deleting your entity instances. Once these stored procedures are defined as functions in the storage schema you can map them in the mapping schema to be used by the ObjectContext for performing inserts, updates and deletes:

```
<EntityTypeMapping TypeName="ContosoModel.Customer">
  <ModificationFunctionMapping>
    <InsertFunction FunctionName="ContosoModel.Store.InsertCustomer">
      <ScalarProperty Name="ID" ParameterName="id" />
      <!-- Other properties mapped to parameters-->
    </InsertFunction>
    <UpdateFunction FunctionName="ContosoModel.Store.UpdateCustomer">
      <ScalarProperty Name="ID" ParameterName="id" Version="Current" />
      <!-- Other properties mapped to parameters-->
    </UpdateFunction>
    <DeleteFunction FunctionName="ContosoModel.Store.DeleteCustomer">
      <!-- Other properties mapped to parameters-->
      <ScalarProperty Name="ID" ParameterName="id" />
    </DeleteFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>
```

The above mapping indicates that we want to use the stored procedures defined as functions in the storage schema to be used for inserts, updates and deletes for the Customer entity.

Using the Entity Client Data Provider

The Entity Client Data Provider is an ADO.NET provider for your ADO.NET Entity Framework. The Entity Client Data Provider is built on top of the ADO.NET provider model. This allows you to work with ADO.NET objects that you are familiar with when working with other ADO.NET provides. For example, we can run a query and retrieve data through a data reader, as follows:

C# Code:

```
EntityConnection con = new EntityConnection();
con.ConnectionString = @"metadata=res://*/Contoso.csdl|res://*/Contoso.ssdl |
res://*/Contoso.msl;provider=System.Data.SqlClient;provider connection
string='Data Source=.\SQLEXPRESS;Initial Catalog=Contoso;Integrated
Security=True;MultipleActiveResultSets=True'";
con.Open();
string sql = "SELECT c.Name,c.City FROM ContosoEntities.Customer as c WHERE
c.Country=@ctry";
EntityCommand cmd = new EntityCommand(sql, con);
cmd.Parameters.Add(new EntityParameter("ctry", DbType.String)).Value =
"Australia";
EntityDataReader rd = cmd.ExecuteReader(CommandBehavior.SequentialAccess);
while (rd.Read())
{
    string name = rd.GetString(0);
    string city = rd.GetString(1);
    //process record
}
```

```
rd.Close();
con.Close();
```

The above code creates an **EntityConnection** object and sets the connection string. The connection string for the Entity Client provider refers to the schema files, the database provider to use, and the connection string to the database. An **EntityCommand** object uses an Entity SQL statement and the **EntityConnection** object to execute a query that returns an **EntityDataReader** instance. The data can be retrieved from the **EntityDataReader** instance.

Summary

The ADO.NET Entity Framework provides a framework and gives life to otherwise static entity data models. Conceptual entities designed in the development life cycle can be mapped to data stores. This allows an application developer to work with an object oriented model, and a database developer to work with a relational model.

Further Reading

- ADO.NET Team Blog: http://blogs.msdn.com/adonet/default.aspx
- Entity SQL reference: http://msdn.microsoft.com/en-us/library/bb387118.aspx

Chapter 10: .NET performance Crib Sheet

If performance is important to an application you are developing, then a 'performance culture' should permeate the whole development process, from initial design to acceptance testing.

This may seem obvious, but a recent high-profile public IT project failed when, just prior to release, it was discovered that each business transaction was taking 27 seconds. It is hard to imagine the sort of developmental chaos that could produce this sort of outcome. Obviously, performance wasn't in the functional spec, and wasn't considered when picking the application architecture.

There are several rules for developing .NET applications that perform well. You should:

- **Design first, then code.** The application's architecture is the largest contributing factor to its performance
- **Have the right tools available to measure performance**, and use them to understand how long the processes take to execute, and why.
- **Write code that is clear and is easy to maintain and understand**. By far the greatest optimisations come through changing the algorithm, so the clearer the code, the easier it is to subsequently optimise. Never indulge in the habit, well-known amongst Perl programmers, of writing obscurantist code from the start, in the mistaken belief that it will run faster at compile time.
- **Gain an understanding of the underlying tasks that the framework performs**, such as memory management and JIT compilation.
- **Set performance goals as early as possible**. You must decide what represents a 'good' performance: perhaps by measuring the speed of the previous version of the application, or of other similar applications. Make sure that clear, documented, performance objectives on all parts of the project are agreed upfront, so that you know when to start and stop optimising, and where the priorities are. Never micro-optimise. You should revise performance goals at each milestone
- **Only optimise when necessary**. Because detailed optimisation is time-consuming, it should be carefully targeted where it will have the most effect. The worst thing a programmer can do is to plough through his code indiscriminately, changing it to optimise its execution. It takes time, makes the code more difficult to understand and maintain, and usually has very little effect.
- **Avoid optimising too early**. Detailed optimisation of code should not be done until the best algorithm is in place and checked for throughput and scalability.
- **Not delay optimisation too far**. The most performance-critical code is often that which is referenced from the most other places in the application: if the fix requires a large change to the way things operate, you can end up having to rewrite or re-factor a huge portion of your application!
- **Assume that poor performance is caused by human error rather than the platform**. It is always tempting to blame the technical platform, but the CLR is intrinsically fast, and is written from the ground up with performance in mind. It is easy to prevent it doing its job by doing things that the design didn't allow for. Even the fastest car slows down if you insist on using the wrong gear.
- **Employ an iterative routine of measuring, investigating, refining/correcting** from the beginning to the end of the product cycle

Measuring and Identifying

There are many aspects to code performance. The most important of these for the developer throughput are: scalability, memory usage, and the startup time of the application. All these must be measured.

The first thing a programmer should do is to find out which part of the code is causing the problem. Usually, the starting place is to use **Performance Counters** and the **CLR Profiler**, using a variety of workloads. Don't make any assumptions about the performance of the platform or of the APIs you are using. You need to measure everything!

This phase of testing and measuring always brings surprises. Other than the Performance Counters and the CLR Profiler, you will want to use a conventional third-party profiler to find out which methods in your application are taking the most time and are being called the most often.

Performance Counters

Performance Counters may be accessed either programmatically or through the Windows Performance Monitor Application (PerfMon). There are Windows performance counters for almost every aspect of the CLR and .NET Framework. They are designed to identify a performance problem by measuring, and displaying, the performance characteristics of a managed application. They are particularly useful for investigating memory management and exception handling.

There are performance counters for almost every aspect of the CLR and .NET Framework. Performance counters will, for example, provide information about:

- The garbage collector.
- The exceptions thrown by the application.
- The application's interaction with COM components, COM+ services, and type libraries.
- The code that has been compiled by the JIT compiler.
- Assemblies, classes, and AppDomains that have been loaded.
- Managed locks and threads used
- Data sent and received over the network
- Remoted objects used by the application.
- The security checks the CLR performs for your application.

These performance counters are always available and aren't invasive; they have low overhead and will not affect the performance characteristics of your application significantly, unless you are monitoring a lot of performance counters.

CLR Profiler

By using the CLR Profiler, you can identify code that allocates too much memory, causes too many garbage collections, and holds on to memory for too long.

The CLR Profiler is designed to show who allocates what on the managed heap, which objects survive on the managed heap, who is holding on to objects, and what the garbage collector does during the lifetime of your application. It is provided as both a command-line application and as a Windows application. With it, you can see much of what is happening to the managed heap of a process whilst a .NET application is running, and can study the behavior of the garbage collector. It allows you to track, and graph, such aspects as object creation, memory consumed by objects, heap size, the type of objects created, and so forth.

The CLR Profiler lets you see which methods allocate which types of objects, and to observe the life-cycle of objects and what keeps objects alive. It logs when methods, classes, and modules get pulled in and called, and by whom.

The CLR Profiler starts the application it will be profiling. It slows the application down as it writes out log data and will quickly fill a large amount of disk space. It is an 'intrusive' tool, in that it causes your application's performance to be between 10 to 100 times slower. If you need timings, you should use the performance metrics.

Writing optimizer-friendly code

At the heart of the .NET Framework is the Common Language Runtime (CLR). The CLR has much in common with the Java Virtual Machine. Java is compiled to Java byte code which then runs in the Java Virtual Machine (JVM). C# code is compiled to an Intermediate Language (IL) which then runs in the Common Language

Runtime (CLR). On the Java platform, byte code is compiled 'Just In Time' (JIT) before being run on the CPU. On the .NET platform, the same happens with the IL code.

Like the JVM, the CLR provides all the runtime services for your code; **Just-In-Time compilation**, **Memory Management** and **Security** are just a few of these services.

When an application is run, the JIT compiler does a number of optimizations such as common sub-expression elimination, de-virtualization, use of intrinsic methods, constant and copy propagation, constant folding, loop unrolling, loop invariant hoisting, Enregistration, Method Inlining, and Range check elimination. Besides the obvious advice of keeping methods short, there is little to do to enhance any of these optimizations except, perhaps, the last three: **Enregistration**, **Method Inlining**, and **Range check elimination**. Although you can improve the effectiveness of compilation, and the speed of the compiled code, you would only consider changing a program in this way when you have already identified a critical section by profiling. You can certainly prevent efficient optimization by doing code obfuscation, because obfuscation can generate code that the optimizer can't deal with, thereby causing a noticeable runtime penalty.

In order to investigate the JIT optimizations, you must look at the assembly code produced by the JIT in run-time, for the release build of your application. If you try to look at the assembly code from within a Visual Studio debug session, you will not see optimized code. This is because JIT optimizations are disabled by default when the process is launched under the debugger. You'll therefore need to attach Visual Studio to a running process, or run the process from within **CorDbg** with the **JitOptimizations** flag on, by issuing the **"mode JitOptimizations 1"** command from within the **CorDbg** command prompt.

Enregistration

The JIT compiler tracks the lifetime of a set number of locals and formal arguments. With this information, it tries to use CPU registers where possible to store locals and method arguments (32-bit integers, object references, 8- and 16-bit integers etc). As there is only a small number of registers, the fewer variables you have the better the chance they will get enregistered, so it makes sense to re-use a variable when possible rather than add a new one.

An 'enum' is normally a 32-bit integer for code generation purposes. Variables which are more than 32-bit in size are never enregistered.

Method Inlining

'In-Line' methods are simpler and therefore faster to execute. The compiler can opt to 'in-line' a method by inserting a child method into the parent method in its entirety. Simple, small methods such as 'Property get and set', which are not virtual, do not initialize private data members, and which do not have exception handling blocks, are good candidates for 'In-Lining'.

An ideal candidate method is one whose compiled code is 32 bytes or less, because no space is wasted; the method pointer occupies 32 or 64 bits anyway and performance is boosted because the CPU doesn't need to 'jump' as often across to different memory locations. Inline methods should not contain any complex branching logic, or any exception handling-related mechanisms, or structs as formal arguments or local variables, or 32-bit floating point arguments or return values.

You cannot 'inline' a virtual method call, because the actual method to be called is unknown at compile-time and can change every time the method is called.

Range Check Elimination

The compiler will generally do range checking of arrays when you are iterating through arrays in a loop using the **for(i=0; i<A.length; i++)** pattern. In certain cases, this is slow and unnecessary. Where you are doing simple, tight loops in code that use just the length of the array, **array.length** as a check for the iterative loop, and use a **localref** rather than static iterator, and you do not cache array or string lengths, the compiler can recognize the fact and will eliminate its own separate check. This can be effective when the loop is scanning large jagged arrays – for instance, as it removes implicit range-checks in both the inner and outer loops.

Coding for performance

Value Types and Reference types

The CLR uses either reference types or value types. Reference types are always allocated on the managed heap and are passed by reference (as the name implies). Value types are allocated on the stack, or inline as part of an object on the heap.

Your code should use value types where a reference type isn't needed, because value types are more economical to use, especially if they are kept small and simple. The value is copied every time you pass it to a method. Be aware, though that if you have large and complex value types, this can get expensive, so some programmers avoid value types altogether unless they are really simple. The cost of performing a garbage collection is a function of the number of live reference types your application has allocated. Value types can also be treated as objects, but they have to be 'Boxed'. Boxing involves the costly operation of allocating a new object on the managed heap and copying the old value into the new object. If you use a method that inherits from **System.Object** on a Value type, then the value is likely to be boxed into a reference type.

Collections

If you create a collection of values, rather than using an array, you will find that every item will be Boxed when added to the collection and unboxed when retrieving the value from the collection. Iterators such as the **foreach** statement may be expensive, as they are unrolled into a sequence of virtual calls. Generic collections such as **ArrayList** are not always a good choice and it is generally better to use custom typed collections instead

A collection should be pre-sized where possible. If you don't specify how many entries your collection is expected to contain, the original default capacity is small and when this value is exceeded, the collection gets resized. Depending on the type of the storage, a new storage object may be allocated, normally at double the capacity, and the contents of the original collection get copied to the newly allocated storage. This takes time and resources, so should be avoided where possible.

Generics

Generic collections provide a way to avoid the boxing and unboxing overhead that comes with using valuetypes in collections. Using generics isn't a free lunch, because they have an effect on the size of the JITed code, particularly if there are a large number of closed constructed types/methods in each generic type/method definition. For best performance, there is an advantage in writing your own optimized collection classes.

Strings

Use the **StringBuilder** class for complex string concatenation and manipulation

Strings are immutable (that is, the referenced text is read-only after the initial allocation); it cannot be modified in place. This provides many performance benefits but is awkward for anyone who is accustomed to C/C++ string manipulation techniques. For example **String.Concat()** is slow and causes unnecessary copies.

If possible, avoid using **foreach()** rather than **for** to enumerate characters in strings.

Minimising start-up times

Much of the work of improving start-up times involves the relatively simple task of loading fewer modules at startup and making sure that nothing is loaded unnecessarily. It also pays to delay initialization by, for example, doing initialisation 'on-demand' when the feature is first used. There is a common misconception that everything

must be loaded and initialized before the application starts, so it is generally easy to improve the application startup times.

The application's configuration settings

Registry settings and INI files present a very quick way of loading application settings. Conversely, using an XML-based config file often isn't a good way to do this, particularly where the application loads more settings than are initially required. XML-based config files are very useful where the configuration is complex and hierarchical, but if it isn't, then it represents an unnecessary overhead.

pre-JITing

Managed Assemblies contain Intermediate Language (IL). The CLR JIT-compiles the IL into optimized CPU instructions. The JIT compilation will take a certain amount of time but this startup time can be improved by doing compilation at install time (pre-JITing) using NGEN.exe. This produces a native binary image for the application, thereby eliminating the JIT overhead, but at the expense of portability. When an NGEN-generated image is run in an incompatible environment, the .NET framework automatically reverts to using JIT. Once NGEN is run against an assembly, the resulting native image is placed into the Native Image Cache for use by all other .NET assemblies. A pre-JITed assembly is a persisted form of a JITed MSIL with a class/v-table layout. It is always worth checking performance before and after using this technique as it can actually have an adverse effect.

If a pre-JITed assembly is not installed in the GAC (Global Assembly Cache), then Fusion (the .NET technology used to locate and load .NET assemblies) needs to verify that the native image and the MSIL assembly have the same version, in order to ensure that the cached native image relates to the current version of the code during binding. To do that, the CLR needs to access pages in the MSIL assembly, which can hurt cold startup time.

Using the Global Assembly Cache

As part of the .NET Framework installation, a central repository is created for storing .NET assemblies. This is called the *Global Assembly Cache*, or GAC. The GAC will have a centralized copy of the .NET Framework itself, and it can also be used to centralize your own assemblies.

If an assembly is installed in the Global Assembly Cache (GAC), it tells the CLR that changes have not occurred to it. This allows it to skip hash verification of the integrity of the assembly, since the check would already have been done in order to place it in the GAC. Otherwise, if an assembly is strongly named, the CLR will check the integrity of the assembly binary on loading, by verifying that the cryptographic hash of the assembly matches the one in the assembly manifest.

It is worth avoiding the hash verification because it is CPU-intensive and involves touching every page in the assembly. The extent of the impact depends on the size of the assembly being verified.

Using Memory Sensibly

Objects in .NET are allocated from the 'managed heap'. The managed heap is so-called because after you ask it for memory the garbage collector takes care of its cleanup once it is no longer required.

Garbage collection in any managed environment begins by assuming all objects are unnecessary until proven otherwise. Essentially, an object proves that it is necessary by its references, or who it is referenced by. If the object is no longer necessary, the object is flagged to be discarded. The process of allocating, managing and disposing of memory comes at a cost. Memory has to be coalesced into contiguous blocks; it must be allocated to the instance of a type; it must be managed over the lifetime of the instance and it must be freed when it is no longer needed.

Memory Allocation

Managed heap allocations do not generally slow code too much. The implicit process is better optimized than the traditional **malloc** routine. Instead of scanning a linked list of memory blocks to find (or coalesce) the first block of sufficient size, it maintains a pointer. Only when memory has to be freed to get such a block does code slow down. If the Garbage collector is confronted with many 'pinned' objects, its task will be slower as it cannot move the memory of pinned objects: the address of the object will have been passed to a native API.

Memory Management

Processing power must be used once memory is allocated to an instance and there is a cost associated with managing memory over the lifetime of an object. The CLR garbage collector is 'generational' in that the managed heap contains 'young' new objects, longer-lived objects, and 'old' objects in separate 'generations'. The Garbage Collector does not always pass over the entire heap before it can recycle memory, so it must check whether objects that have memory assigned to them still hold references to objects in younger 'generations'. The Garbage collector will always collect the smallest section of the heap possible in order to free enough memory for the application to continue.

A scan that just covers 'young' new objects (Generation 0) is very rapid, whereas a scan that takes in old objects (Generation 2) is likely to affect your application's performance.

Memory assigned to a live object can be moved around in order to meet the demands for a new memory block. If an object is too large for this sort of treatment, it is allocated to a special area of the heap called the **Large Object Heap**, where objects are not relocated. When the garbage collector reaches the Large Object Heap, it forces a full collection. The rapid creation and disposal of large (approx >80K) object instances is likely to slow an application down.

The more objects that are created, the more allocations take place. The greater the number of allocations, the more overhead is involved in Garbage collection. For purely performance reasons, rather than logical design reasons, it is best to stick to two types of objects:

- Short-term objects that can contain object references to other short-term objects, or
- Long term objects with references only to other long-term objects.

Large short-term objects slow applications down, as do long-term objects that refer to young objects.

It is all a matter of compromise, but this may need attention if the *'Memory: % Time in GC'* performance counter goes above 30%. The ratio between the *'Memory: # Gen 0 Collections'* and *'Memory: # Gen 2 Collections'* performance counters will give the best indication as to whether your 'allocation profile' is likely to cause the Garbage collection to slow the application down.

In the CLR, there are two different Garbage Collectors, a **Workstation GC** and a **Server GC**. The Workstation GC is optimized for low latency. It works on one thread only and therefore on one CPU only, whereas the Server GC (used by ASP.NET, for example) scales better for larger, multiprocessor applications.

Finalizing

Finalizable objects have to be maintained in the system longer than objects without finalizers. Finalization is the process whereby a dead object's native resources, such as database connections or operating system handles, are freed up if it has not been disposed. This happens before its memory is returned to the heap. Because it can happen in the background, it is queued-up (to the **Finalizable** Queue). The Finalization process only starts when the Garbage collector comes across the object. The object is passed to a different (**FReachable**) queue and the object is promoted to the next generation. The finalization process is then done by a different thread. After the object is finalized, the Garbage collector can free up and reuse the memory.

If your object does not require finalization, do not implement it, because there is an overhead to calling the Finalize method. If your object needs finalization, then implement the **Dispose** pattern. Finalization is usually

only needed when an object that requires cleanup is not deliberately killed. Don't make the parent class 'finalizable'. Only the wrapper class around the unmanaged objects that need cleanup needs to be finalizable. Make these 'Finalizable' objects as small and simple as possible. They should never block.

Disposing

If you no longer need an object, then it can be 'disposed of'. Disposing of the object is relatively easy and well optimized in the CLR. In both VB.NET and C# you'd generally employ a **using** block. You need to have your object implement the **IDisposable** interface and provide an implementation for the **Dispose** method. In the **Dispose** method, you will call the same cleanup code that is in the Finalizer and inform the GC that it no longer needs to finalize the object, by calling the **GC.SuppressFinalization** method.

If you have a class with a Finalizer, it makes sense to use a **Dispose** method to cut down the overhead of the finalization. You should call just one common finalization, or cleanup, function for both the **Dispose** method and the Finalizer, to save on the maintenance overhead. Where a **Close** method would be more logical than a **Dispose** method, then the **Close** method that you write can simply call the **Dispose** method.

Weak References

If an object has a reference to it on the stack, in a register, in another object, or in one of the other GC Roots, by a 'strong reference', then the object will not be recycled by the Garbage Collector because it assumes it is still required. Unless you particularly want this, then use Weak References.

Common Language Runtime issues

Exception Handling

Exceptions represent a very efficient way of handling errors; far better than the old VB **On Error Goto**. However, they should only be used in exceptional or unexpected circumstances. Do not use exceptions for normal flow control: It is a common mistake in Java, VB.NET and C#, to use exception-handling instead of checking for all the conditions that might cause an error. The reason for avoiding use of exceptions in normal flow control is the processing required once an exception is thrown. An expensive stack walk is required to find an appropriate exception handler for the thrown exception.

Security

If code is Fully Trusted, and the security policy remains at the default, then security has a negligible additional impact on the throughput and startup time of a .NET application. If code is Partially Trusted, or MyComputer Grant Set is narrowed, then the effect on the application will increase.

Changes to policy, such as lowering grants to the MyComputer zone or exclusive code groups, can affect performance and startup time.

When the CLR finds an Authenticode signature when it loads a strongly named assembly into a process in the .NET Framework 2.0, it generates Publisher Evidence from it. This means that it has to fully validate the certificate by contacting the issuing authority across the internet to ensure the certificate has not been revoked. Applications can sometimes experience very long startup times in these cases. If the internet connection is not there, the process times-out after a long delay. In such circumstances, the application must be able to turn off auto-generation of publisher evidence, or use authenticode certificates that don't have a Certificate Revocation List Distribution Point (CDP) embedded in them. Sometimes the application developer is unaware that he is using a third-party component that he is using a third-party component that requires a validated certificate that has to be authenticated at startup.

Late-Binding

Late-binding is generally the slowest reflection technique. Occasionally it is necessary to invoke the members of a class dynamically, or to dynamically emit, JIT and execute a method. Though at times it is very fast, this facility often brings with it a performance tradeoff. You may be using reflection, such as late-binding, without realizing it: an API that you use might, in turn, make transitive use of the reflection APIs.

If, in Visual BasicVB.NET and JScript.NET, you use a variable without explicitly declaring it, then Reflection is used to convert the object to the correct type at runtime. Unfortunately, a late-bound call is a great deal slower than a direct call. If you are using VB.NET and you don't require late binding, you can make the compiler enforce the declaration by strongly typing your variables and turning off implicit casting. Simply include the **`Option Explicit On`** and **`Option Strict On`** at the top of your source files.

COM Interop and Platform Invoke

COM Interop and Platform Invoke are easy to use but they are significantly slower than regular managed calls. There is a fixed delay of around fifty instructions to make the transition between native and managed code, and a variable delay caused by the task of marshalling any arguments and return values. This delay depends entirely on the nature of the transformation. In some cases, the conversion of data types, such as the conversion of a string between CLR Unicode and Win32 ANSI, can be a surprisingly slow, iterative process, whereas primitive types, and arrays of primitive types, are almost free.

Where there is no alternative to Interop or P/Invoke, the number of transformations should be kept to a minimum. Code that is continuously passing data back and forth will inevitably be slow. As much work as possible should be performed inside each Interop call, so as to avoid multiple, frequent invocations. It is also sensible to copy data just once into the unmanaged world, and maintain that as a static resource.

Initialize managed threads to the appropriate COM threading model. Avoid implementing **IDispatch** for managed COM servers, and calling unmanaged COM servers through **IDispatch**.

Threading and Synchronization

Threading and synchronisation is handled by the CLR. It may seem obvious to use threading to increase the performance of an application and, certainly, multithreading can significantly increase the perceived (and sometimes the real) performance of a GUI application. However, it can occasionally have the reverse effect because of the overhead involved. It generally pays to use the thread pool and minimize the creation of threads. Avoid fine-grained locks and the use of shared locks **(RWLock**) for general locking.

Use asynchronous calls for improved scalability on multiprocessor machines and for improved perceived performance in Windows Forms applications.

Reflection

Reflection, which is a way of fetching type information at runtime, relies on getting at the metadata that is embedded in managed assemblies. Many reflection APIs require searching and parsing of the metadata, which takes time.

Type comparison operations take less time than Member enumerations. The latter allow you to inspect the methods, properties, fields, events, constructors and so on, of a class. At design time, they are essential, but at runtime they can cause delays, if done indiscriminately, due to the overheads associated with reflection.

Type comparisons (e.g. **typeof()**), member enumerations (e.g. Type.**GetFields()**) and member access (e.g. **Type.InvokeMember()**)– all will use reflection and so should be avoided in performance-critical code. It is not always obvious when reflection is being used: some APIs use reflection as a side effect (e.g: **Object.ToString()**).

Conclusions

If you read this crib sheet in search of 'golden' performance rules, then prepare to be disappointed. The most important golden rule is that there aren't many of them. Tips that make sense in one version of .NET are often irrelevant in another. The .NET framework is being subjected to an enormous effort to improve its performance, due to competitive pressure from Java. Many of the snippets of advice in this crib sheet may not be relevant or even true for your particular application. You need to be suspicious of advice and should instead measure, profile, test and retest under a number of different test loads and settings.

Optimization is time-consuming and unexciting. By continuously profiling and measuring, you can avoid a lot of unnecessary re-factoring, and make your code easier to read for others as well.

Essential tools

- **Perfmon**
- **CLR Profiler**
- Reflector.net Lutz Roeder's Reflector is the class browser, explorer, analyzer and documentation viewer for .NET. Reflector allows you to easily view, navigate, search, decompile and analyze .NET assemblies in C#, Visual Basic and IL. It is best thought of as a sort of ILDASM++. There is an excellent 'Call Tree', that can be used to drill down into IL methods to see what other methods they call.
- **ANTS Profiler**. This code and memory profiler identifies performance bottlenecks in any .NET application. It provides you with quantitative data such as line-level code timings and hit count, allowing you to go straight down to the line of code responsible for performance inefficiencies, and optimize efficiently. (It's so much easier to make an algorithm run more efficiently when you know exactly where to focus your performance-boosting work)
- **SQL Data Generator** This is ideal for checking code scalability during unit-testing. See Using SQL Data Generator with your Unit Tests for a step-by-step instructions.

Handy References:

Articles

- Performance Considerations for Run-Time Technologies in the .NET Framework
- Performance Tips and Tricks in .NET Applications
- Writing High-Performance Managed Applications : A Primer
- CLR Inside Out- Improving Application Startup Time
- Improving String Handling Performance in .NET Framework Applications
- NET Compact Framework version 2.0 Performance and Working Set FAQ
- Authenticode and Assemblies

Newsgroup

- microsoft.public.dotnet.framework.performance