1

SQL Server 2012 Architecture

WHAT'S IN THIS CHAPTER

- New Important Features in SQL Server 2012
- How New Features Relate to Data Professionals Based on Their Role
- SQL Server Architecture Overview
- Editions of SQL Server and How They Affect the Data Professional

SQL Server 2012 offers a fresh look at how organizations and their developers, information workers, and executives use and integrate data. A tremendous number of new features and improvements focus on extending SQL Server more into SharePoint, improving self-service options, and increasing data visualization, development, monitoring and exploration capabilities. This chapter is not a deep dive into the architecture but provides enough information to give you an understanding of how SQL Server operates.

SQL SERVER 2012 ECOSYSTEM

This thing called SQL Server has become quite large over the past few releases. This first section provides a review of the overall SQL Server ecosystem, which is now referred to as less of a product and more of an ecosystem, because there are so many interactions with other products and features that drive increased performance, scale, and usability. Following are three major areas of focus for the release of SQL Server 2012:

- **Performance:** Features such as improved core support, columnstore indexes, compression enhancements, and Always On make this the most powerful, available release of SQL Server.
- **Self Service:** With new data exploration tools such as Power View, improvements in SQL Azure Business Intelligence (BI,), data quality and master data offerings, and

PowerPivot for SharePoint enable users to be closer to the data at all times and to seek and deliver intelligence more rapidly than ever.

- **Integration and collaboration:** New integrations for reporting services, PowerPivot, and claims authentication in SharePoint 2010 provide a strong foundation for the significant focus on self-service in this release. The new BI semantic model approach extends into the cloud as well with reporting services now in SQL Azure and more features promised to come.

NEW IMPORTANT FEATURES IN 2012

There are a number of new things that you will be excited about, depending on your role and how you use SQL Server. This section touches on the features you should be checking out and getting your hands on. Many of these features are quick to get up and running, which is exciting for those readers who want to begin delivering impact right away.

Production DBA

Production DBAs are a company's insurance policy that the production database won't go down. If the database does go down, the company cashes in its insurance policy in exchange for a recovered database. The Production DBA also ensures that the server performs optimally and promotes database changes from development to quality assurance (QA) to production. New features include the following:

- **AlwaysOn**: Availability functionality including availability groups and the ability to file over databases in groups that mimic applications. This includes new readable secondary servers, a big enhancement.
- **FileTable**: Additional file-based data storage
- **Extended Events**: A new functionality built into SQL Server 2012 that provides lightweight and extensive tracing capability
- Improved functionality and stability in SQL Server Management Studio (now in Visual Studio 2010 shell)
- Distributed replay capabilities
- Improved debugging functionality including expression support and breakpoint validation.
- Columnstore indexes for optimizing large data volumes
- Improved statistics algorithm for very large databases
- Improved compression and partitioning capabilities

Development DBA

Since the release of SQL Server 2000, there has been a trend away from full-time Production DBAs, and the role has merged with that of the Development DBA. The trend may have slowed, though, with laws such as Sarbanes-Oxley, which require a separation of power between the person developing

the change and the person implementing the change. In a large organization, a Production DBA may fall into the operations department, which consists of the network of administrators and Windows-support administrators. In other instances, a Production DBA may be placed in a development group. This removes the separation of power that is sometimes needed for regulatory reasons.

Development DBAs play a traditional role in an organization. They wear more of a developer's hat and are the development staff's database experts and representatives. This administrator ensures that all stored procedures are optimally written and that the database is modeled correctly, both physically and logically. The development DBA also may be the person who writes the migration processes to upgrade the database from one release to the next. The Development DBA typically does not receive calls at 2:00 A.M like the Production DBA might for failed backups or similar problems. Things development DBAs should be excited about in this new release include the following:

- New TSQL and spatial functionality
- SQL Server data tools: A new TSQL development environment integrated with Visual Studio
- New DAX expression language that provides Excel-like usability with the power of multidimensional capabilities
- New tabular model for Analysis Services: Provides in-memory OLAP capabilities in a quick time to value format

The Development DBA typically reports to the development group and receives requests from a business analyst or another developer. In a traditional sense, Development DBAs should never have modification access to a production database. They should, however, have read-only access to the production database to debug in a time of escalation.

Business Intelligence DBA and Developer

The Business Intelligence (BI) DBA is a new role that has evolved due to the increased capabilities of SQL Server. In SQL Server 2012, BI grew to be an incredibly important feature set that many businesses could not live without. The BI DBA or developer is an expert at these features. This release is a treasure trove of new BI functionality including new enhancements to Reporting Services Integration, data exploration tools such as Power View, and a dramatic set of enhancements that make PowerPivot easier and more accessible than ever. Additionally, the new Tabular model in SSAS delivers the ability to create new PowerPivot-like "in memory" BI projects to SharePoint for mass user consumption.

Development BI DBAs specialize in the best practices, optimization, and use of the BI toolset. In a small organization, a Development BI DBA may create your SSIS packages to perform Extract Transform and Load (ETL) processes or reports for users. In a large organization, developers create the SSIS packages and SSRS reports. The Development BI DBA is consulted regarding the physical implementation of the SSIS packages and Analysis Services (SSAS) cubes. Development BI DBAs may be responsible for the following types of functions:

- Model\consult regarding Analysis Services cubes and solutions
- Create reports using Reporting Services
- Create\consult around ETL using Integration Services
- Develop deployment packages to be sent to the Production DBA

These responsibilities, coupled with these following new features make for an exciting time for the BI-oriented folks:

- Rapid data discovery with Power View and PowerPivot
- Managed Self-Service BI with SharePoint and BI Semantic Model
- Credible, consistent data with Data Quality Services and Master Data Management capabilities
- Robust DW solutions with Parallel Data Warehouse and Reference Architectures

SQL SERVER ARCHITECTURE

Many people just use SQL Server for its classic use: to store data. This release of SQL Server focuses on expanding the capabilities that were introduced in SQL Server 2008 R2, which was largely a self-service business intelligence and SharePoint feature release. The additional functionality in SQL Server 2012 not only enables but encourages users to go beyond simply storing data in SQL Server; this release can now be the center of an entire data strategy. New tools such as Power View and PowerPivot quickly integrate on top of SQL Server and can provide an easy user interface (UI) for SQL Server and other systems' data. This section covers the primary file types in SQL Server 2012, file management, SQL Client, and system databases. It also covers an overview of schemas, synonyms, and Dynamic Management Objects. Finally, it also goes into the new SQL Server 2012 data types.

Database Files and Transaction Log

The architecture of database and transaction log files remains relatively unchanged from prior releases. Database files serve two primary purposes depending on their type. Data files hold the data, indexes, and other data support structure within the database. Log files hold the data from committed transactions to ensure consistency in the database.

Database Files

A database may consist of multiple filegroups. Each filegroup must contain one or more physical data files. Filegroups ease administrative tasks for a collection of files. Data files are divided into 8KB data pages, which are part of 64KB extents. You can specify how full each data page should be with the fill factor option of the `create/alter index` T-SQL command. In SQL Server 2012 Enterprise Edition, you continue to have the capability to bring your database partially online if a single file is corrupt. In this instance, the DBA can bring the remaining files online for reading and writing, and users receive an error if they try to access the other parts of the database that are offline.

In SQL 2000 and before, the largest row you could write was 8060 bytes. The exceptions to this limit are `text`, `ntext`, `image`, `varchar(max)`, `varbinary(max)`, and `nvarchar(max)` columns, which may each be up to 2 gigabytes and are managed separately. Beginning with SQL 2005, the 8KB limit applies only to those columns of fixed length. The sum of fixed-length columns and pointers for other column types must still be less than 8060 bytes per row. However, each variable-length column may be up to 8KB in size allowing for a total row size of well over 8060 bytes. If your actual row size

exceeds 8060 bytes, you may experience some performance degradation because the logical row must now be split across multiple physical 8060-byte rows.

Transaction Log

The purpose of the transaction log is to ensure that all committed transactions are persisted in the database and can be recovered, either through rollback or point in time recovery. The transaction log is a *write-ahead log*. As you make changes to a database in SQL Server, the data is written to the log, and then the pages that need to be changed are loaded into memory (specifically into the write buffer portion of the buffer pool). The pages are then dirtied by having the changes written to them. Upon checkpoint, the dirty pages are written to disk, making then now clean pages which no longer need to be part of the write buffer. This is why you may see your transaction log grow significantly in the middle of a long-running transaction even if your recovery model is set to simple. (Chapter 17, "Backup and Recovery" covers this in much more detail.)

SQL Native Client

The SQL Native Client is a data-access method that shipped with SQL Server 2005 and was enhanced in 2012 and is used by both OLE DB and ODBC for accessing SQL Server. The SQL Native Client simplifies access to SQL Server by combining the OLE DB and ODBC libraries into a single access method. The access type exposes these features in SQL Server:

- Database mirroring
- Always On readable secondary routing
- Multiple Active Result Sets (MARS)
- Snapshot isolation
- Query notification
- XML data type support
- User-defined data types (UDTs)
- Encryption
- Performing asynchronous operations
- Using large value types
- Performing bulk copy operations
- Table-value parameters
- Large CLR user-defined types
- Password expiration

In these features, you can use the feature in other data layers such as Microsoft Data Access Components (MDAC), but it takes more work. MDAC still exists, and you can use it if you don't need some of the new functionality of SQL Server 2008\2012. If you develop a COM-based application, you should use SQL Native Client; and if you develop a managed code application

like in C#, you should consider using the .NET Framework Data Provider for SQL Server, which is robust and includes the SQL Server 2008\2012 features as well.

Standard System Databases

The system databases in SQL Server are crucial, and you should leave them alone most of the time. The only exceptions to that rule is the `model` database, which enables you to deploy a change such as a stored procedure to any new database created, and `tempdb`, which may need to be altered to help with scaling your workload. The following sections go through the standard system databases in detail.

If certain system databases are tampered with or corrupted, you run the risk that SQL Server will not start. The master database contains all the stored procedures and tables needed for SQL Server to remain online.

The Resource Database

SQL Server 2005 added the `Resource` database. This database contains all the read-only critical system tables, metadata, and stored procedures that SQL Server needs to run. It does not contain any information about your instance or your databases because it is written to only during an installation of a new service pack. The `Resource` database contains all the physical tables and stored procedures referenced logically by other databases. You can find the database by default in `C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Binn`, and there is only one `Resource` database per instance.

The use of drive C: in the path assumes a standard setup. If your machine is set up differently, you may need to change the path to match your setup. In addition, the `.MSSQLSERVER` *is the instance name. If your instance name is different, use your instance name in the path.*

In SQL Server 2000, when you upgraded to a new service pack, you needed to run many long scripts to drop and re-create system objects. This process took a long time to run and created an environment that couldn't be rolled back to the previous release after the service pack. In SQL Server 2012, when you upgrade to a new service pack or hot fix, a copy of the `Resource` database overwrites the old database. This enables you to both quickly upgrade your SQL Server catalog and roll back a release.

The `Resource` database cannot be seen through Management Studio and should never be altered unless you're under instruction to do so by Microsoft Product Support Services (PSS). You can connect to the database under certain single-user mode conditions by typing the command **`USE MSSQLSystemResource`**. Typically, a DBA runs simple queries against it while connected to any database, instead of having to connect to the resource database directly. Microsoft provides some

functions that enable this access. For example, if you were to run this query while connected to any database, it would return your `Resource` database's version and the last time it was upgraded:

```
SELECT serverproperty('resourceversion') ResourceDBVersion,
serverproperty('resourcelastupdatedatetime') LastUpdateDate
```

Do not place the `Resource` *database on an encrypted or compressed drive. Doing this may cause upgrade or performance issues.*

The master Database

The `master` database contains the metadata about your databases (database configuration and file location), logins, and configuration information about the instance. You can see some of the metadata stored in `master` by running the following query, which returns information about the databases that exist on the server:

```
SELECT * FROM sys.databases
```

The main difference between the `Resource` and `master` databases is that the `master` database holds data specific to your instance, whereas the `Resource` database just holds the schema and stored procedures needed to run your instance but does not contain any data specific to your instance.

You should rarely create objects in the `master` *database. If you create objects here, you may need to make frequent master db backups.*

tempdb Database

The `tempdb` database is similar to the operating system paging file. It's used to hold temporary objects created by users, temporary objects needed by the database engine, and row-version information. The `tempdb` database is created each time you restart SQL Server. The database will be re-created to its original database size when the SQL Server is started. Because the database is re-created each time, you cannot back it up. Data changes made to objects in the `tempdb` database benefit from reduced logging. You must have enough space allocated to your `tempdb` database because many operations that you use in your database applications use the `tempdb`. Generally speaking, you should set `tempdb` to autogrow as it needs space. Typically your `tempdb` size varies but you should understand how much temp space your application will use at peak and make sure there is enough space with 15–20 percent overhead for growth. If there is not enough space, the user may receive one of the following errors:

- **1101** or **1105:** The session connecting to SQL Server must allocate space in `tempdb`.
- **3959:** The version store is full.
- **3967:** The version store must shrink because `tempdb` is full.

model Database

`model` is a system database that serves as a template when SQL Server creates a new database. As each database is created, SQL Server copies the `model` database as the new database. The only time this does not apply is when you restore or attach a database from a different server.

If a table, stored procedure, or database option should be included in each new database that you create on a server, you may simplify the process by creating the object in `model`. When the new database is created, `model` is copied as the new database, including the special objects or database settings you have added to the `model` database. If you add your own objects to `model`, it should be included in your backups, or you should maintain a script that includes the changes.

msdb Database

`msdb` is a system database that contains information used by SQL Server agent, log shipping, SSIS, and the backup and restore system for the relational database engine. The database stores all the information about jobs, operators, alerts, and job history. Because it contains this important system-level data, you should back up this database regularly.

Schemas

Schemas enable you to group database objects together. You may want to do this for ease of administration because you can apply security to all objects within a schema. Another reason to use schemas is to organize objects so that the consumers may find the objects they need easily. For example, you may create a schema called `HumanResource` and place all your employee tables and stored procedures into it. You could then apply security policies on the schema to allow appropriate access to the objects contained within it.

When you refer to an object, you should always use the two-part name. The `dbo` schema is the default schema for a database. An `Employee` table in the `dbo` schema is referred to as `dbo.Employee`. Table names must be unique within a schema. You could create another table called `Employee` in the `HumanResources` schema. It would be referred to as `HumanResources.Employee`. This table actually exists in the AdventureWorks sample database for SQL Server 2012. (All SQL Server 2012 samples must be downloaded and installed separately from `wrox.com`.) A sample query using the two-part name follows:

```
SELECT BusinessEntityID, JobTitle
FROM HumanResources.Employee
```

Prior to SQL 2005, the first part of the two-part name was the user name of the object owner. The problem with that implementation was related to maintenance. If a user who owned objects were to leave the company, you could not remove that user login from SQL Server until you ensured that all the objects owned by the user were changed to a different owner. All the code that referred to the objects had to be changed to refer to the new owner. By separating ownership from the schema name, SQL 2005 through 2012 removes this maintenance problem.

Synonyms

A *synonym* is an alias, or alternative name, for an object. This creates an abstraction layer between the database object and the consumer. This abstraction layer enables you to change some of the physical implementation and isolate those changes from the consumer. The following example is

related to the use of linked servers. You may have tables on a different server that need to be joined to tables on a local server. You refer to objects on another server using the four-part name, as shown in the following code:

```
SELECT Column1, Column2
FROM LinkedServerName.DatabaseName.SchemaName.TableName
```

For example, you might create a synonym for `LinkedServerName.DatabaseName.SchemaName.Tablename` called `SchemaName.SynonymName`. Data consumers would refer to the object using the following query:

```
SELECT Column1, Column2
FROM SchemaName.SynonymName
```

This abstraction layer now enables you to change the location of the table to another server, using a different linked server name, or even to replicate the data to the local server for better performance without requiring any changes to the code that refers to the table.

A synonym cannot reference another synonym. The `object_id` *function returns the id of the synonym, not the id of the related base object. If you need column-level abstraction, use a view instead.*

Dynamic Management Objects

Dynamic Management Objects (DMOs) and functions return information about your SQL Server instance and the operating system. DMO's are grouped into two different classes: dynamic management views (DMVs) and dynamic management functions (DMFs). DMV's and DMF's simplify access to data and expose new information that was not available in versions of SQL Server prior to 2005. DMOs can provide you with various types of information, from data about the I/O subsystem and RAM to information about Service Broker.

Whenever you start an instance, SQL Server begins saving server-state and diagnostic information in memory which DMV's and DMF's can access. When you stop and start the instance, the information is flushed from the views and fresh data begins to be collected. You can query the views just like any other table in SQL Server with the two-part qualifier. For example, the following query uses the `sys.dm_exec_sessions` DMV to retrieve the number of sessions connected to the instance, grouped by login name:

```
SELECT login_name, COUNT(session_id) as NumberSessions
FROM sys.dm_exec_sessions GROUP BY login_name
```

Some DMFs are functions that accept parameters. For example, the following code uses the `sys.dm_io_virtual_file_stats` dynamic management function to retrieve the I/O statistics for the AdventureWorks data file:

```
USE AdventureWorks
GO
SELECT * FROM
```

```
sys.dm_io_virtual_file_stats(DB_ID('AdventureWorks'),
FILE_ID('AdventureWorks_Data'))
```

Many new DMV's and DMF's exist in SQL Server 2012. These views focus on improved insight into new and existing areas of functionality and include the following:

- AlwaysOn Availability Groups Dynamic Management Views and Functions
- Change Data Capture Related Dynamic Management Views
- Change Tracking Related Dynamic Management Views
- Common Language Runtime Related Dynamic Management Views
- Database Mirroring Related Dynamic Management Views
- Database-Related Dynamic Management Views
- Execution-Related Dynamic Management Views and Functions
- SQL Server Extended Events Dynamic Management Views
- FileStream and FileTable Dynamic Management Views
- Full-Text Search and Semantic Search Dynamic Management Views and Functions
- Index-Related Dynamic Management Views and Functions
- I/O-Related Dynamic Management Views and Functions
- Object-Related Dynamic Management Views and Functions
- Query Notifications Related Dynamic Management Views
- Replication-Related Dynamic Management Views
- Resource Governor Related Dynamic Management Views
- Security-Related Dynamic Management Views and Functions
- Server-Related Dynamic Management Views and Functions
- Service Broker Related Dynamic Management Views
- Spatial Data Related Dynamic Management Views and Functions
- SQL Server Operating System Related Dynamic Management Views
- Transaction-Related Dynamic Management Views and Functions

SQL Server 2012 Data Types

Data types are the foundation of table creation in SQL Server. As you create a table, you must assign a data type for each column. This section covers some of the more commonly used data types in SQL Server. Even if you create a custom data type, it must be based on a standard SQL Server data type. For example, you may create a custom data type (`Address`) by using the following syntax, but notice that it based on the SQL Server standard `varchar` data type:

```
CREATE TYPE Address
FROM varchar(35) NOT NULL
```

If you change the data type of a column in a large table in SQL Server Management Studio's table designer interface, the operation may take a long time. You can observe the reason for this by scripting the change from the Management Studio interface. Management Studio creates a secondary temporary table with a name such as `tmpTableName` and then copies the data into the table. Finally, the interface deletes the old table and renames the new table with the new data type. Other steps along the way, of course, handle indexes and any relationships in the table.

If you have a large table with millions of records, this process can take more than 10 minutes, and in some cases more than 1 hour. To avoid this, you can use a simple one-line T-SQL statement in the query window to change the column's data type. For example, to change the data type of the Job `Title` column in the `Employees` table to a `varchar(70)`, you could use the following syntax:

```
ALTER TABLE HumanResources.Employee ALTER COLUMN JobTitle Varchar(70)
```

When you convert to a data type that may be incompatible with your data, you may lose important data. For example, if you convert from a numeric data type that has data such as 15.415 to an integer, the number 15.415 would be rounded to a whole number.

You may want to write a report against your SQL Server tables that displays the data type of each column inside the table. You can do this dozens of ways, but a popular method shown in the following example joins the `sys.objects` table with the `sys.columns` table. There are two functions that you may not be familiar with in the following code. The `TYPE_NAME()` function translates the data type id into its proper name. To go the opposite direction, you could use the `TYPE_ID()` function. The other function of note is `SCHEMA_ID()`, which is used to return the identity value for the schema. This is useful primarily when you want to write reports against the SQL Server metadata.

```
USE AdventureWorks
GO
SELECT o.name AS ObjectName,
        c.name AS ColumnName,
        TYPE_NAME(c.user_type_id) as DataType
FROM sys.objects o
JOIN sys.columns c
ON o.object_id = c.object_id
WHERE  o.name ='Department'
and o.Schema_ID = SCHEMA_ID('HumanResources')
```

This code returns the following results (the ***Name*** data type is a user-defined type):

```
ObjectName           ColumnName     DataType
 ----------------------------------------------------

Department           DepartmentID   smallint
Department           Name           Name
Department           GroupName      Name
Department           ModifiedDate   datetime
```

Character Data Types

Character data types include `varchar`, `char`, `text`, and. This set of data types stores character data. The primary difference between the `varchar` and `char` types is data padding. If you have a column called `FirstName` that is a `varchar(20)` data type and you store the value of "Brian" in the column, only 5 bytes are physically stored plus a little overhead. If you store the same value in a `char(20)` data type, all 20 bytes would be used. SQL inserts trailing spaces to fill the 20 characters.

You might ask yourself, if you want to conserve space, why would you ever use a `char` *data type? There is a slight overhead to using a* `varchar` *data type. If you store a two-letter state abbreviation, for example, you're better off using a* `char(2)` *column. Although some DBAs have opinions about this that border on religious conviction, generally speaking it's good to find a threshold in your organization and specify that anything below this size becomes a* `char` *versus a* `varchar`*. A good guideline is that, in general, any column less than or equal to 5 bytes should be stored as a* `char` *data type instead of a* `varchar` *data type, depending on your application needs. Beyond that point, the benefit of using a* `varchar` *begins to outweigh the cost of the overhead.*

The `nvarchar` and `nchar` data types operate the same way as their `varchar` and `char` counterparts, except these data types can handle international Unicode characters. This comes at a cost, though. Data stored as Unicode consumes 2 bytes per character. If you were to store the value of "Brian" in an `nvarchar` column, it would use 10 bytes, and storing it as an `nchar(20)` would use 40 bytes. Because of this overhead and added space, you should not use Unicode columns unless you have a business or language need for them. Think ahead to the future and consider instances in which you might need them. If there is no future business need, avoid them.

Table 1-1 shows the data types with short descriptions and the amount of storage required.

TABLE 1-1: SQL Server Data Types

DATA TYPE	DESCRIPTION	STORAGE SPACE
`Char(n)`	N between 1 and 8,000 characters	n bytes
`Nchar(n)`	N between 1 and 4,000 Unicode characters	(2 x n bytes)
`Nvarchar(max)`	Up to ((2 to the 30th power) – 1) (1,073,741,823) Unicode characters	2 x characters stored + 2 bytes overhead
`Text`	Up to ((2 to the 31st power) – 1) (2,147,483,647) characters	1 byte per character stored +2bytes overhead
`Varchar(n)`	N between 1 and 8,000 characters	1 byte per character stored + 2 bytes overhead
`Varchar(max)`	Up to ((2 to the 31st power) – 1) (2,147,483,647) characters	1 byte per character stored + 2 bytes overhead

Exact Numeric Data Types

Numeric data types consist of `bit`, `tinyint`, `smallint`, `int`, `bigint`, `numeric`, `decimal`, `money`, `float`, and `real`. Each of these data types stores different types of numeric values. The first data type, `bit`, stores only a null, 0 or a 1, which in most applications translates into true or false. Using the `bit` data type is perfect for on and off flags, and it occupies only a single byte of space. Table 1-2 shows other common numeric data types.

TABLE 1.2: Exact Numeric Data Types

DATA TYPE	DESCRIPTION	STORAGE SPACE
`bit`	0, 1, or Null	1 byte for each 8 columns of this data type
`tinyint`	Whole numbers from 0 to 255	1 bytes
`smallint`	Whole numbers from –32,768 to 32,767	2 bytes
`int`	Whole numbers from –2,147,483,648 to 2,147,483,647	4 bytes
`bigint`	Whole numbers from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 bytes
`numeric(p,s)` or `decimal(p,s)`	Numbers from –1,038 + 1 through 1,038 –1	Up to 17 bytes
`money`	–922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
`smallmoney`	–214,748.3648 to 214,748.3647	4 bytes

Numeric data types, such as `decimal` and `numeric` can store a variable number of digits to the right and left of the decimal place. *Scale* refers to the number of digits to the right of the decimal. *Precision* defines the total number of digits, including the digits to the right of the decimal place. For example, 14.88531 would be a `numeric(7,5)` or `decimal(7,5)`. If you were to insert 14.25 into a `numeric(5,1)` column, it would be rounded to 14.3.

Approximate Numeric Data Types

The data types `float` and `real` are included in this group. They should be used when floating-point data must be represented. However, because they are approximate, not all values can be represented exactly.

The `n` in the `float(n)` is the number of bits used to store the mantissa of the number. SQL Server uses only two values for this field. If you specify between 1 and 24, SQL uses 24. If you specify between 25 and 53, SQL uses 53. The default is 53 when you specify `float()`, with nothing in parenthesis.

Table 1-3 shows the approximate numeric data types with a short description and the amount of storage required.

TABLE 1-3: Approximate Numeric Data Types

DATA TYPE	DESCRIPTION	STORAGE SPACE
`float[(n)]`	–1.79E+308 to –2.23E-308,0, 2.23E-308 to 1.79E+308	N<=24 – 4 bytes N> 24– 8 bytes
`real()`	–3.40E+38 to –1.18E-38,0, 1.18E-38 to 3.40E+38	4 bytes

The synonym for `real` *is* `float(24)`.

Binary Data Types

Binary data types such as `varbinary`, `binary`, `varbinary(max`store binary data such as graphic files, Word documents, or MP3 files. The values are hexadecimal 0x0 to 0xf. The `image` data type stores up to 2GB outside the data page. The preferred alternative to an `image` data type is the `varbinary(max)`, which can hold more than 8KB of binary data and generally performs slightly better than an `image` data type. New in SQL Server 2012 is the capability to store `varbinary(max)` objects in operating system files via FileStream storage options. This option stores the data as files and is not subject to the 2GB size limit of `varbinary(max)`.

Table 1-4 shows the binary data types with a short description and the amount of storage required.

TABLE 1-4: Binary Data Types

DATA TYPE	DESCRIPTION	STORAGE SPACE
`Binary(n)`	N between 1 and 8,000 hex digits	n bytes
`Varbinary(n)`	N between 1 and 8,000 hex digits	1 byte per character stored + 2 bytes overhead
`Varbinary(max)`	Up to 231 – 1(2,147,483,647) characters	1 byte per character stored + 2 bytes overhead

Date and Time Data Types

The `datetime` and `smalldatetime` types both store date and time data. The `smalldatetime` is 4 bytes and stores from January 1, 1900, through June 6, 2079, and is accurate to the nearest minute.

The `datetime` data type is 8 bytes and stores from January 1, 1753, through December 31, 9999, to the nearest 3.33 millisecond.

SQL Server 2012 has four new date-related data types: `datetime2`, `dateoffset`, `date`, and `time`. You can find examples using these data types in SQL Server Books Online.

The `datetime2` data type is an extension of the `datetime` data type, with a wider range of dates. Time is always stored with hours, minutes, and seconds. You can define the `datetime2` data type with a variable parameter at the end — for example, `datetime2(3)`. The `3` in the preceding expression means to store fractions of seconds to three digits of precision, or .999. Valid values are between 0 and 7, with a default of 3.

The `datetimeoffset` data type is just like the `datetime2` data type, with the addition of the time offset. The time offset is + or – up to 14 hours, and contains the UTC offset so that you can rationalize times captured in different time zones.

The `date` data type stores the date only, a long-requested piece of functionality. Alternatively, the `time` data type stores the time only. The `time` data type also supports the `time(n)` declaration, so you can control granularity of the fractional seconds. As with `datetime2` and `datetimeoffset`, `n` can be between 0 and 7.

Table 1-5 shows the date/time data types with a short description and the amount of storage required.

TABLE 1-5: Date and Time Data Types

DATA TYPE	DESCRIPTION	STORAGE SPACE
`Date`	January 1, 1 to December 31, 9999	3 bytes
`Datetime`	January 1, 1753 to December 31, 9999, Accurate to nearest 3.33 millisecond	8 bytes
`Datetime2(n)`	January 1, 1 to December 31, 9999 N between 0 and 7 specifies fractional seconds	6 to 8 bytes
`Datetimeoffset(n)`	January 1, 1 to December 31, 9999 N between 0 and 7 specifies fractional seconds +- offset	8 to 10 bytes
`SmalldateTime`	January 1, 1900 to June 6, 2079, Accurate to 1 minute	4 bytes
`Time(n)`	Hours:minutes:seconds.9999999 N between 0 and 7 specifies fractional seconds	3 to 5 bytes

Other System Data Types

Table 1-6 shows several other data types, which you have not yet seen.

TABLE 1-6: Other System Data Types

DATA TYPE	DESCRIPTION	STORAGE SPACE
`Cursor`	This contains a reference to a cursor and may be used only as a variable or stored procedure parameter.	Not applicable
`Hierarchyid`	Contains a reference to a location in a hierarchy	1 to 892 bytes + 2 bytes overhead
`SQL_Variant`	May contain the value of any system data type except `text`, `ntext`, `image`, `timestamp`, `xml`, `varchar(max)`, `nvarchar(max)`, `varbinary(max)`, and user-defined data types. The maximum size allowed is 8,000 bytes of data + 16 bytes of metadata.	8,016 bytes
`Table`	Used to store a data set for further processing. The definition is like a Create Table. Primarily used to return the result set of a table-valued function, they can also be used in stored procedures and batches.	Dependent on table definition and number of rows stored
`Timestamp or Rowversion`	Unique per table, automatically stored value. Generally used for version stamping, the value is automatically changed on insert and with each update.	8 bytes
`Uniqueidentifier`	Can contain Globally Unique Identifier (GUID). `guid` values may be obtained from the `Newsequentialid()` function. This function returns values that are unique across all computers. Although stored as a binary 16, it is displayed as a `char(36)`.	16 bytes
`XML`	is Unicode by definition	Up to 2GB

A `cursor` *data type may not be used in a* `Create Table` *statement.*

The `XML` data type stores an XML document or fragment. It is stored like an `nvarchar(max)` in size depending on the use of UTF-16 or UTF-8 in the document. The `XML` data type enables the use of special constructs for searching and indexing. (This is covered in more detail in Chapter 15, "Replication".)

CLR Integration

In SQL Server 2012, you can also create your own data types, functions, and stored procedures using the portion of the Common Language Runtime referred to as (SQLCLR). This enables you to write more complex data types to meet your business needs in Visual Basic or C#, for example. These types are defined as a class structure in the base CLR language.

EDITIONS OF SQL SERVER

SQL Server 2012 is available in several editions, and the features available to you in each edition vary widely. The editions you can install on your workstation or server also vary based on the operating system. The editions of SQL Server range from SQL Express on the lowest end to Enterprise Edition on the highest.

Edition Overview

There are three major editions of SQL Server 2012. There are additional, smaller editions typically not recommended for production environments so they are not covered here. For more details on these, see `www.microsoft.com/sqlserver`. Figures 1-1 and 1-2 describe the new editions releasing with SQL Server 2012.

FIGURE 1-1

SQL Server 2012 offers three principal SKUs:

- **Enterprise Edition** contains all the new SQL Server 2012 capabilities, including high availability and performance updates that make this a mission critical-ready database. This edition contains all the BI functionality as well.
- **Business Intelligence Edition** is new in SQL Server 2012. The BI Server Edition offers the full suite of powerful BI capabilities in SQL Server 2012, including PowerPivot and Power View. One major focus of this edition is empowering end users with BI functionality. This is ideal for projects that need advanced BI capabilities but don't require the full OLTP performance and scalability of Enterprise Edition. The new BI Edition is inclusive of the standard edition and still contains the basic OLTP offerings.
- **Standard Edition** remains as it is today, designed for departmental use with limited scale. It has basic database functionality and basic BI functionality. New features are now included in Standard such as compression.

Figure 1-2 provides you with an at-a-glance view of some of the key features in SQL Server 2012 by edition.

SQL Server Capabilities	SQL Server Editions		
	Standard	Business Intelligence	Enterprise
Maximum Number of Cores	16 cores	16 cores – DB OS Max – BI	OS Max
Basic OLTP	●	●	●
Basic Reporting & Analytics	●	●	●
Programmability & Developer Tools	●	●	●
Manageability (Management studio, policy based management)	●	●	●
Enterprise Data Management (Data quality, master data services)		●	●
Self-service Business Intelligence (Power View, PowerPivot for SPS)		●	●
Corporate Business Intelligence (Semantic model, advanced analytics)		●	●
Advanced Security (Advanced auditing, transparent data encryption)			●
Data Warehousing (Column store index, compression, partitioning)			●
Maximum Scalability and Performance			●
High Availability	Limited	Basic	●
AlwaysOn			●
Virtualization Licensing	1 VM	1 VM	Unlimited w/ SA

FIGURE 1-2

The basic functionality is kept in the Standard Edition. Key enterprise BI features are included in the BI SKU. When you need to go to high end data warehousing and enterprise level high availability, the Enterprise Edition is the right edition for you.

Figure 1-2 provides a more detailed look at the SQL Server 2012 Editions and related capabilities. As you can see, for the Standard Edition, there is a 16 core maximum. For the Business Intelligence Edition, there is a 20 core maximum for database use and up to the OS Maximum for BI. The Enterprise Edition can be used up the maximum number of cores in the operating system.

Both the BI Edition and Enterprise Edition offer the full premium BI capabilities of SQL Server 2012, including Enterprise Data Management, Self-Server BI, and Corporate BI features. Enterprise Edition adds mission critical and Tier 1 database functionality with the maximum scalability, performance, and high availability. With the Enterprise Edition under Software Assurance, customers also get unlimited virtualization, with the ability to license unlimited virtual machines.

Licensing

SQL Server 2012 has significant licensing changes. This new licensing scheme can impact your environment if you do not run Software Assurance on those licenses. This section is an overview of the changes, not an exhaustive licensing discussion. See your Microsoft account manager for more details on these changes.

SQL Server 2012 pricing and licensing better aligns to the way customers purchase database and BI products. The new pricing and licensing offers a range of benefits for customers including the following:

- **SQL Server 2012 offers market leading Total Cost of Ownership (TCO):**
  - SQL Server 2012 continues to be the clear high-value leader among major vendors. This is exemplified through the pricing model; features included in non-enterprise editions and world class support no matter the edition.
  - Customers with Software Assurance get significant benefits and ways to help ease the transition to new licensing models. These benefits include access to upgrades and enhanced support options during your license term.
  - Customers with Enterprise Agreements have the easiest transition to SQL Server 2012 and realize the greatest cost-savings.
- **SQL Server 2012 is cloud-optimized:**
  - SQL Server 2012 is the most virtualization-friendly database with expanded virtualization licensing, the flexibility to license per VM, and excellent support for Hyper-V.
  - Customers also have the ability to support hybrid scenarios that span on-premises, private cloud, and public clouds through better integration with SQL Azure.
- **SQL Server 2012 pricing and licensing is designed to enable customers to pay as they grow:**
  - The new streamlined editions are aligned to database and BI needs.
  - For datacenter scenarios, licensing is better aligned to hardware capacity.
  - For BI, licensing is aligned to user-based access, which is the way most customers are accustomed to purchasing BI.

CPU Core (Not Processor) Licenses

With the release of SQL Server 2012, Microsoft switched to a per core processing model. Don't fret because for most of your servers the costs should stay the same. The CPU Core licenses (available only for the Standard and Enterprise Edition) are sold in two core "packs." So a quad core CPU needs two of these packs per socket. These license packs cost half of what a SQL Server 2008 R2 CPU license cost. The catch here is that you must purchase at least 4 cores per CPU.

For example:

- Two sockets with 2 cores each, you need 4 license "packs" (8 core licenses).
- Two sockets with 4 cores each, you need 4 license "packs" (8 core licenses).
- Two sockets with 6 cores each, you need 6 license "packs" (12 core licenses).
- Two sockets with 8 cores each, you need 8 license "packs" (16 core licenses).

Virtualized SQL Server and Host Based Licensing

When you run a virtualized SQL Server, you must license at least four cores for the VM. If you have more than four virtual CPUs on the VM, you must have a CPU Core license for each virtual CPU that you have assigned to the VM.

SQL Server 2012 still includes host-based licensing for those customers with Software Assurance and an Enterprise Agreement. The host-based licensing works just like it did before: you purchase enough Enterprise Edition CPU Core licenses for the host, and you can run as many virtual machines running SQL Server as you want. This will likely be the preferred way for many of you. For those customers not running with Software Assurance or an Enterprise Agreement, they will want to contact your Microsoft Representative or reseller since host based licensing is not available for those customers.

As you can see, this is a lot of change. The pricing has changed too but is not discussed here due to the wide variations depending on your individual agreements with Microsoft. Microsoft has tried hard to make sure the cost does not go up dramatically for many of you, so don't fret, but be judicious and check this out with your Microsoft Account teams.

SUMMARY

The architecture for SQL Server 2012 has advancements under the covers that will improve performance, increase developer efficiency and system availability, and decrease overall operating cost. It is important to focus on your current and future roles and understand the types of features and editions that will apply to your situation and organizational needs.