Instructor Notes:

Instructor Notes:

Lesson Objectives

- In this lesson, you will learn:
  - Working with Data Types
  - Working with SQL Server Schemas
  - Working with DDL, DML ,DCL statements
  - Implementing Data Integrity

Lesson Objectives

Designing, creating and maintaining tables are one of the very important tasks a database developer performs while maintaining databases for any enterprise. In this particular lesson, you walk through these and other tasks related to tables. You also take a look at the Data Types available in SQL Server 2012. You will be also introduced to implementing declarative data integrity in your tables.

Instructor Notes:

Introduction to Data Types in SQL Server

- SQL Server has an extensive list of data types to choose from, including some that are new to respected versions.
- In SQL Server, each column, local variable, expression and parameter has a related data type
- A data type is an element that specifies the kind of data that the item can hold
- SQL Server introduces new additions of data types that can simplify your database development code
- You can also define your own data types in SQL Server
- These User Defined Data Types (UTDs)/ Alias Data Types are based on the system-supplied data types

Introduction to Data Types in SQL Server

SQL Server 2012 has an extensive list of data types to choose from, including some that are new to SQL Server 2012. While working with SQL server data, a developer extensively works with objects like table column, local variable, expression, parameter and so on.
At the same time before you can create a table, you must define the data types for the data that will be stored in that table. Data types specify the type of information (characters, numbers, or dates) that a column can hold, as well as how the data is stored. SQL Server 2012 supplies more than 30 specific system data types. It also supports alias data types, which are user-defined data types based on system data types.
You should take into consideration following important attributes while assigning data type to an Object.

The type of data contained by the Object
The length or size of the stored value
The precision of the number (Numeric Data Types only)
The scale of the number (Numeric Data Types only)

Instructor Notes:

What are System-Supplied Data Types?

- Integers (whole number)
  - int -2^31 to 2^31-1
  - smallint -2^15 to 2^15 -1
  - bigint -2^63 to 2^63-1
  - tinyint 0 to 255
  - bit 1 or 0
- numeric (Fixed precision)
  - decimal -10^38 +1 to 10^38 –1
  - numeric Equivalent to decimal
- Approximate numeric
  - float -1.79E + 308 to 1.79E + 308
  - real -3.40E + 38 to 3.40E + 38

Transact-SQL has these base data types :

Exact Numerics

Integers

bigint - Integer (whole number) data from -9223372036854775808 through 223372036854775807.
int - Integer (whole number) data from -2,147,483,648 through 2,147,483,647.
smallint - Integer data from 2^15 (-32,768) through 2^15 - 1 (32,767).
tinyint - Integer data from 0 through 255.
bit - Integer data with either a 1 or 0 value.

decimal and numeric
decimal - Fixed precision and scale numeric data from -10^38 +1 through 10^38 –1.
numeric - Functionally equivalent to decimal.

Approximate Numeric
float - Floating precision number data from -1.79E + 308 through 1.79E + 308.
real - Floating precision number data from -3.40E + 38 through 3.40E + 38.

Instructor Notes:

What are System-Supplied Data Types?

- Monetary (with accuracy to a ten-thousandth of a monetary unit)
  - money -2^63 to 2^63 - 1
  - smallmoney -214,748.3648 through +214,748.3647
- Date and Time
  - datetime
  - smalldatetime
- Character (s)
  - char Fixed-length non-Unicode character data
  - varchar Variable-length non-Unicode data (max length 8,000)

money and smallmoney
money - Monetary data values from -922,337,203,685,477.5808 through +922,337,203,685,477.5807, with accuracy to a ten-thousandth of a monetary unit.
smallmoney - Monetary data values from -214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit.

datetime and smalldatetime
datetime - Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds. range, a larger default fractional precision, and optional user-specified precision.
smalldatetime - Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.

Character Strings
char - Fixed-length non-Unicode character data with a maximum length of 8,000 characters.
varchar -Variable-length non-Unicode data with a maximum of 8,000 characters.

Instructor Notes:

Instructor should talk that image and text are larger data types and their storage is separate from normal char and binary data types ,

The instructor is expected to talk about encoding schemes and also talk about a l little on ASCII , MBCS and Unicode etc .

The instructor also has to explain that Unicode data type is twice the size of non Unicode data type .

Instructor can also explain in which scenarios one uses the Unicode data type

What are System-Supplied Data Types?

- Binary
  - Binary — Fixed-length binary data ,max of 8000 bytes
  - Varbinary — Varying length binary data ,max of 8000 bytes
- Large Objects
  - Image
  - Text
- Unicode Data type storage is two times the byte size
  - Nchar
  - Nvarchar, nvarchar(max)
  - ntext

Binary Strings
binary - Fixed-length binary data with a maximum length of 8,000 bytes.
varbinary - Variable-length binary data with a maximum length of 8,000 bytes.

Large Objects
text - Variable-length non-Unicode data with a maximum length of 2^31 - 1 (2,147,483,647) characters.
image - Variable-length binary data with a maximum length of 2^31 - 1 (2,147,483,647) bytes.

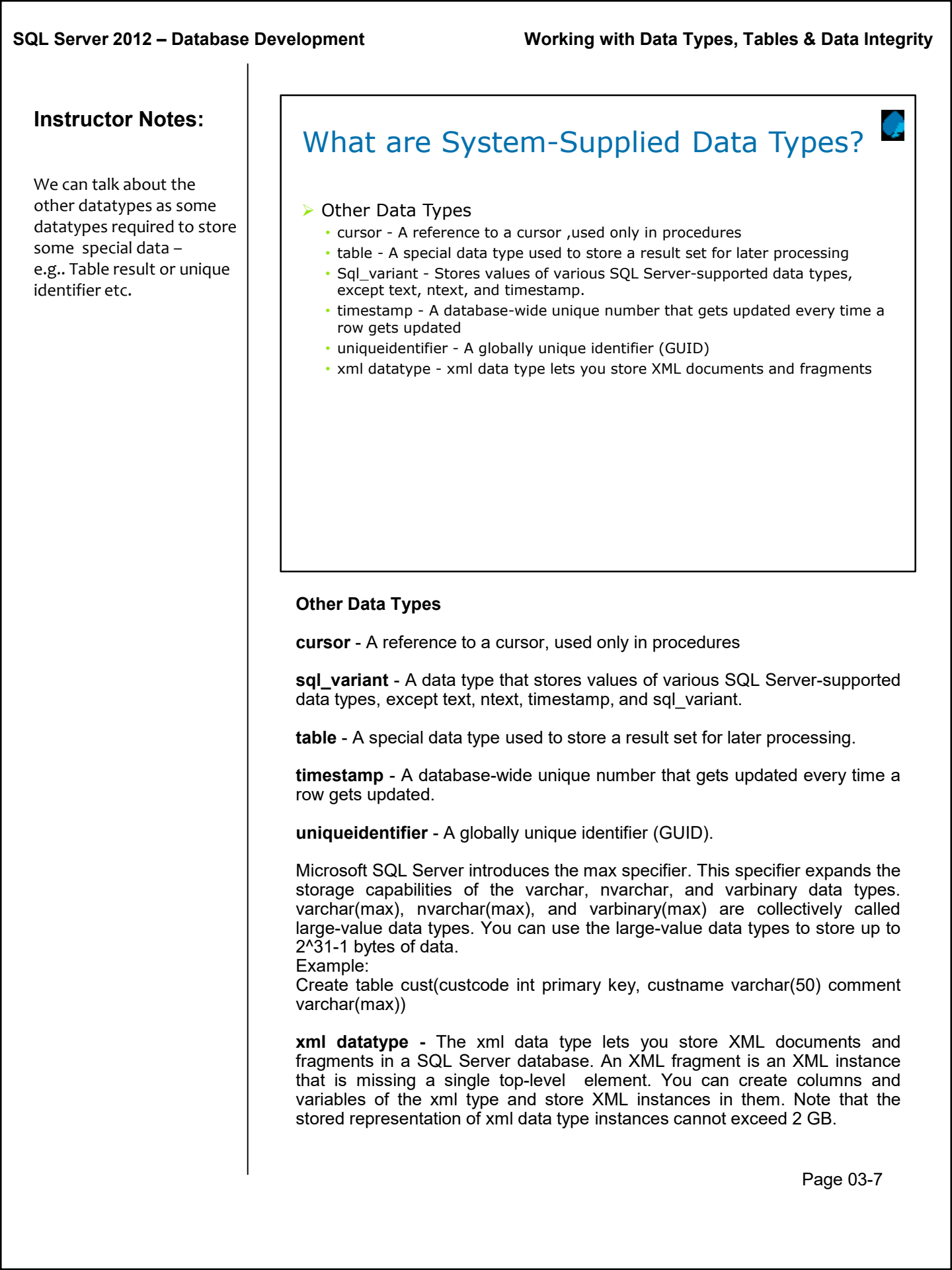
Unicode Data types
Unicode is a character encoding system for representing characters from different language like traditional Chinese, kanji etc. .

Unicode provides a unique number for every character. The numbers occupy two bytes. The unicode data type supported in SQL Server are

Nchar - Fixed-length Unicode data with a maximum length of 4,000 characters.
Nvarchar - Variable-length Unicode data with a maximum length of 4,000 characters.
nvarchar(max) - Variable-length Unicode data with a maximum length of 230 characters
ntext - Variable-length Unicode data with a maximum length of 1,073,741,823 characters.

Instructor Notes:

We can talk about the other datatypes as some datatypes required to store some special data – e.g.. Table result or unique identifier etc.



Other Data Types

cursor - A reference to a cursor, used only in procedures

sql_variant - A data type that stores values of various SQL Server-supported data types, except text, ntext, timestamp, and sql_variant.

table - A special data type used to store a result set for later processing.

timestamp - A database-wide unique number that gets updated every time a row gets updated.

uniqueidentifier - A globally unique identifier (GUID).

Microsoft SQL Server introduces the max specifier. This specifier expands the storage capabilities of the varchar, nvarchar, and varbinary data types. varchar(max), nvarchar(max), and varbinary(max) are collectively called large-value data types. You can use the large-value data types to store up to 2^31-1 bytes of data.
Example:
Create table cust(custcode int primary key, custname varchar(50) comment varchar(max))

xml datatype - The xml data type lets you store XML documents and fragments in a SQL Server database. An XML fragment is an XML instance that is missing a single top-level element. You can create columns and variables of the xml type and store XML instances in them. Note that the stored representation of xml data type instances cannot exceed 2 GB.

Instructor Notes:

What are System-Supplied Data Types?

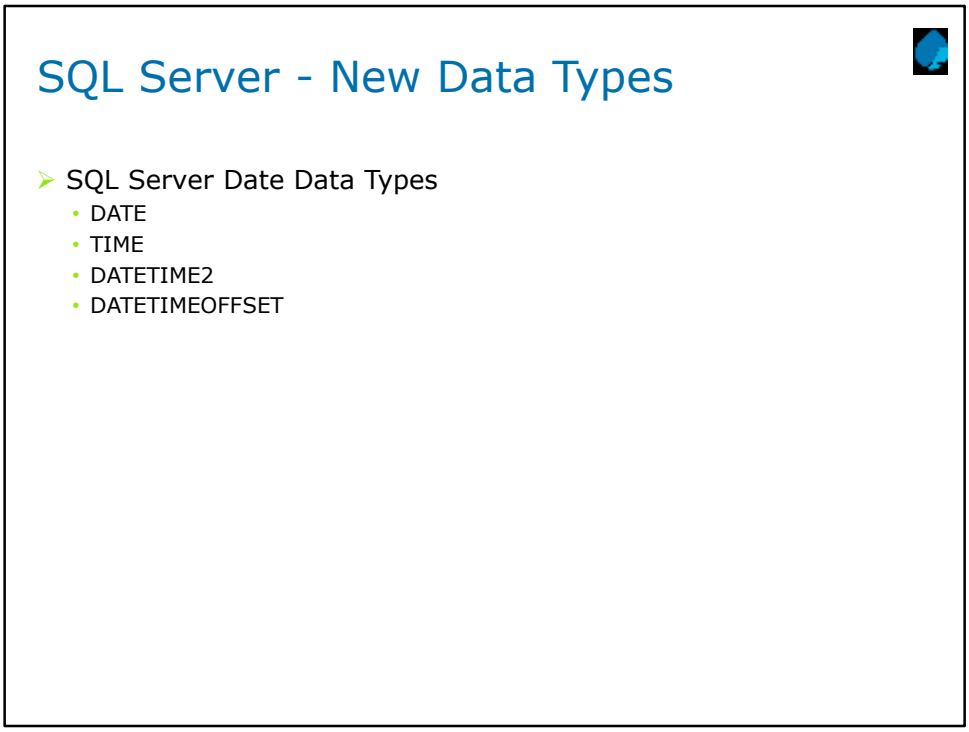
- Large value data types:
  - varchar(max)
  - nvarchar(max)
  - varbinary(max)
- For non-unicode data MAX allows up to 2^31 -1 bytes
- For unicode data MAX allows up to 2^30 -1 bytes
- XML
  - To store data in XML Format
  - XML DML for performing insert, update and deletes on the XML based columns

You can optionally associate an XML schema collection with a column, a parameter, or a variable of the xml data type. The schemas in the collection are used to validate and type the XML instances. In this case, the XML is said to be typed.

The xml data type and associated methods help integrate XML into the relational framework of SQL Server.

Example:
CREATE TABLE T1(Col1 int primary key, Col2 xml)

Instructor Notes:



SQL Server Date Data Types

Previous versions of SQL Server offered two date data types: datetime and smalldatetime. These data types were sufficient for most applications, but could be cumbersome in certain cases. For instance, both data types have a date and time portion, which is great if that's what you want, but cumbersome if all you need to store is just the date or just the time.
Similarly, the date ranges imposed by these two data types - 1753-01-01 to 9999-12-31 for datetime and 1900-01-01 to 2079-06-06 for smalldatetime - are insufficient for a small percentage of applications. SQL Server 2008 remedies these ills by keeping the datetime and smalldatetime and introducing four new date data types: time, date, datetime2, and datetimeoffset.

As you can probably guess by their names, the time and date data types track just a time and just a date portion, respectively. The datetime2 data type is like the datetime data type, but has a larger window of dates (from 0001-01-01 to 9999-12-31) and greater precision on the time portion.

The datetimeoffset data type has the same range and precision as the datetime2 data type, but enables an offset from UTC to be specified, which makes it easier to record and display times respective to the end user's timezone.

Instructor Notes:

SQL Server - Date Data Types

Data Type	Format	Range	Storage Size (bytes)
time	hh:mm:ss[.nnnnnnn]	00:00:00.0000000 through 23:59:59.9999999	5
date	YYYY-MM-DD	0001-01-01 through 9999-12-31	3 bytes, fixed
smalldatetime	YYYY-MM-DD hh:mm:ss	1900-01-01 through 2079-06-06	4 bytes, fixed.
datetime	YYYY-MM-DD hh:mm:ss[.nnn]	1753-01-01 through 9999-12-31	8 bytes
datetime2	YYYY-MM-DD hh:mm:ss[.nnnnnnn]	0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999	6 bytes for precisions less than 3; 7 bytes for precisions 3 and 4. All other precisions require 8 bytes.
datetimeoffset	YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+|-]hh:mm	0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 (in UTC)	10 bytes, fixed is the default with the default of 100ns fractional second precision

TIME

The datatype TIME is primarily used for storing the time of a day. This includes Hours, minutes, Seconds etc. It is based on a 24-hour clock. The datatype TIME can store seconds up to the fraction of 9999999.

Syntax : time [(fractional second precision)]

Usage : DECLARE @MyTime time(7)

CREATE TABLE Table1 (Column1 time(7))

DATE

This data type is useful to store the dates without the time part, we can store dates starting from 00001-01-01 through 9999-12-31 i.e. January 1, 1 A.D. through December 31, 9999 A.D. It supports the Gregorian Calendar and uses 3 bytes to store the date.

Syntax: date

Usage: DECLARE @MyDate date

CREATE TABLE Table1 (Column1 date)

Instructor Notes:

DATETIME2

This is a new data type introduced in SQL Server and this date/time data type is introduced to store the high precision date and time data. The data type can be defined for variable lengths depending on the requirement. This data type also follows the Gregorian Calendar. The Time Zone can't be specified in this data type. This is still useful because it gives you a complete flexibility to store the date time data as per your requirement..

Syntax: datetime2 [(fractional seconds precision)]

Usage: DECLARE @MyDatetime2 datetime2(7)

CREATE TABLE Table1 (Column1 datetime2(7))

SMALLDATETIME and DATETIME

Microsoft SQL Server continues to support existing data types such as datetime and smalldatetime.

SMALLDATETIME: Defines a date that is combined with a time of day. The time is based on a 24-hour day, with seconds always zero (:00) and without fractional seconds. The date range for the datatype smalldatetime is from 1900-01-01 through 2079-06-06 and time range supported is 00:00:00 through 23:59:59.

Syntax: smalldatetime

Usage: DECLARE @MySmalldatetime smalldatetime

CREATE TABLE Table1 (Column1 smalldatetime)

DATETIME: This data type Defines a date that is combined with a time of day with fractional seconds that is based on a 24-hour clock. The date range for the datatype datetime is from January 1, 1753, through December 31, 9999 and time range supported is 00:00:00 through 23:59:59.997.

Syntax: datetime

Usage: DECLARE @MyDatetime datetime

CREATE TABLE Table1 (Column1 datetime)

DATETIMEOFFSET

This is the new data type that is included in SQL Server and this data type is the most advanced in the league. We can store high precision date/ time with the Date Time Offset. We can't store the Time Zone like Eastern Time, Central Time etc. in the data type but can store the offset -5:00 for EST and -6:00 CST and so on. The data type is not Day light saving aware.

The date range is between 0001-01-01 and 9999-12-31 or January 1,1 A.D. through December 31, 9999 A.D. and the Time Range is between 00:00:00 and 23:59:59.9999999. The offset range is between -14:00 through +14:00. The precision of the data type can be set manually and it follows the Gregorian Calendar.

Usage: datetimeoffset [(fractional seconds precision)]

Usage: DECLARE @MyDatetimeoffset datetimeoffset(7)

CREATE TABLE Table1 (Column1 datetimeoffset(7))

Instructor Notes:

Examples:

DATE & TIME Data Type

```
Use AdventureWorks
Go

Create Schema Trade
Go

CREATE TABLE Trade.BankTran
(
    TransID          BIGINT IDENTITY(1,1) PRIMARY KEY ,
    AccountNo        INT,
    BankTranType     VARCHAR(50),
    TransDate        DATE,
    TransTime        Time(7)
)
Go

INSERT INTO Trade.BankTran
(AccountNo,BankTranType,TransDate,TransTime)
Values(101,'Transfer',sysdatetime(),sysdatetime())
Go
INSERT INTO Trade.BankTran
(AccountNo,BankTranType,TransDate,TransTime)
Values(102,'Transfer',sysdatetime(),sysdatetime())
Go

SELECT * FROM Trade.BankTran
```

Result:

```
TransID    AccountNo   BankTranType TransDate  TransTime
-------------------- ----------- ----------------------------------------------
1                  101         Transfer         2012-04-22 18:13:59.4657106
2                  102         Transfer         2012-04-22 18:13:59.4807114

(2 row(s) affected)
```

Instructor Notes:

Examples:

Comparison between DATETIME2 & DATETIME data type with example:

```
USE AdventureWorks
GO

CREATE TABLE  TimeTable
(
    FirstDate DATETIME,
    LastDate  DATETIME2(4)
)
GO

DECLARE @Interval INT
SET @Interval = 1000
WHILE (@Interval > 0)
BEGIN
                INSERT TimeTable (FirstDate, LastDate)
                VALUES (SYSDATETIME(), SYSDATETIME())
SET @Interval = @Interval - 1
END
GO

SELECT * FROM TimeTable
GO
```

Output:

```
FirstDate                   LastDate
-------------------------   -------------------------
2012-04-22 19:54:52.700  2012-04-22 19:54:52.69
2012-04-22 19:54:52.703  2012-04-22 19:54:52.70
2012-04-22 19:54:52.703  2012-04-22 19:54:52.70
2012-04-22 19:54:52.707  2012-04-22 19:54:52.70
2012-04-22 19:54:52.707  2012-04-22 19:54:52.70
2012-04-22 19:54:52.707  2012-04-22 19:54:52.70
2012-04-22 19:54:52.710  2012-04-22 19:54:52.70
2012-04-22 19:54:52.710  2012-04-22 19:54:52.70
2012-04-22 19:54:52.713  2012-04-22 19:54:52.71
2012-04-22 19:54:52.713  2012-04-22 19:54:52.71
2012-04-22 19:54:52.717  2012-04-22 19:54:52.71
```

Note that when using the datetime data type is rounded to increments of .000, .003, or .007 seconds. However the datetime2 has a larger date range, a larger default fractional precision, and optional user-specified precision.

Instructor Notes:

Examples:

DATETIMEOFFSET:

```
SELECT
  CAST(SYSDATETIMEOFFSET() AS datetimeoffset(7)) AS
     'datetimeoffset'
```

Output:

```
datetimeoffset
----------------------------------
2012-04-22 20:28:16.1262803 +05:30

(1 row(s) affected)
```

SYSDATETIMEOFFSET(): This function returns a datetimeoffset(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is included.

NOTE – IST, which is GMT plus 5.30 hrs, came into existence in 1905.

Instructor Notes:

Spatial Data Type – What is Spatial Data?

- Information about the location and shape of a geometric object:
  - Store locations
  - Sales regions
  - Customer sites
  - Area within a specific distance of a location
- Two types:
  - Planar (or Euclidean) data for coordinate points on a flat, bounded surface. Distances are measured directly between points
  - Geodetic (or ellipsoidal) data for latitude and longitude points on the surface of the Earth. Distances are measured taking into account the curvature of the ellipsoidal surface

What is Spatial Data?

Spatial data is information that stores the location and shape of geometric objects, for example, a building, a road, a river, a city, a continent, or an ocean.

The object can be identified by a single point or a set of points that create a geometric shape. Spatial data is rapidly becoming more important to many industries. You can use it to locate stores, identify sales regions, pinpoint customer sites, and generate areas that are local to a location.

There are two types of spatial data: planar and geodetic. Planar, or Euclidean, spatial data views the Earth as if it is projected onto a planar surface like a map. Geodetic, or ellipsoidal, data views the Earth as an ellipsoid and works with information such as latitude and longitude coordinates.

The key difference between planar and ellipsoidal data is the way that operations on the data are performed. For example, distances between planar points are measured directly between the points by using simple trigonometry, whereas distances between geodetic data points take into account the curvature of the ellipsoidal surface.

Instructor Notes:

The geometry and geography Data Types

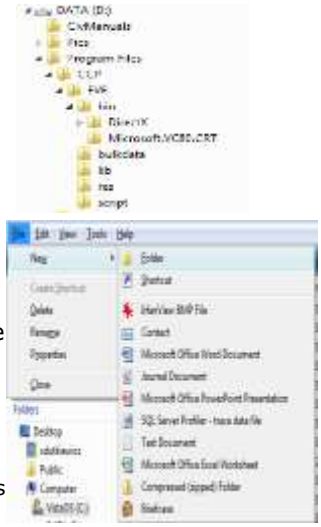
- SQL Server supports two spatial data types:
  - geometry for planar spatial data
  - geography for ellipsoidal spatial data
- Both data types:
  - Are implemented as .NET Framework common language runtime types
  - Can store points, lines, and areas
  - Provide members to perform spatial operations
- Common uses:
  - Geometry - localized geospatial data such as street maps
  - Geography - locations on the Earth's surface and integration
  - geospatial systems

Types of Spatial Data

There are two types of spatial data. The **geometry** data type supports planar, or Euclidean (flat-earth), data. The geometry data type conforms to the Open Geospatial Consortium (OGC) Simple Features for SQL Specification version 1.1.0.
In addition, SQL Server supports the **geography** data type, which stores ellipsoidal (round-earth) data, such as GPS latitude and longitude coordinates.

Instructor Notes:

SQL Server 2008 – Hierarchyid Data Types



- SQL Server 2008 introduces a new system-provided data type "Hierarchyid" to encapsulate hierarchical relationships
- We can use hierarchyid as a data type to create tables with a hierarchical structure
- It is designed to store values that represent the position of nodes of a hierarchal tree structure
- This type is internally stored as a VARBINARY value
- Examples where the hierarchyid type makes it easier to store and query hierarchical data include the following:
  - Organizational structures
  - A set of tasks that make up a larger projects (like a GANTT chart)
  - File systems (folders and their sub-folders)
  - A graphical representation of links between web pages

Hierarchyid

While hierarchical tree structures are commonly used in many applications, SQL Server has not made it easy to represent and store them in relational tables. In SQL Server 2008, the HIERARCHYID data type has been added to help resolve this problem. It is designed to store values that represent the position of nodes of a hierarchal tree structure.

The new HIERARCHYID data type in SQL Server 2008 is a system-supplied CLR UDT that can be useful for storing and manipulating hierarchies. This type is internally stored as a VARBINARY value that represents the position of the current node in the hierarchy (both in terms of parent-child position and position among siblings).

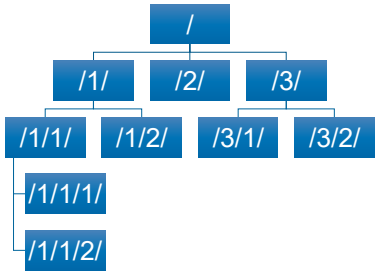
Unlike standard data types, the HIERARCHYID data type is a CLR user-defined type, and it exposes many methods that allow you to manipulate the date stored within it. For example, there are methods to get the current hierarchy level, get the previous level, get the next level, and many more. In fact, the HIERARCHYID data type is only used to store hierarchical data; it does not automatically represent a hierarchical structure. It is the responsibility of the application to create and assign HIERARCHYID values in a way that represents the desired relationship. Think of a HIERARCHYID data type as a place to store positional nodes of a tree structure, not as a way to create the tree structure.

The HIERARCHYID data type makes it easier to express these types of relationships without requiring multiple parent/child tables and complex joins.

Instructor Notes:

Hierarchyid – Key Properties & Limitations

- Key Properties
  - Extremely compact
  - Comparison is in depth-first order
  - Support for arbitrary insertions and deletions
- Limitations
  - Does not automatically represent a tree
  - No guarantee that hierarchyid values in a column would be unique
  - Hierarchical relationships represented by hierarchyid values are not enforced like a foreign key relationship



Key Properties of hierarchyid

A value of the hierarchyid data type represents a position in a tree hierarchy. Values for hierarchyid have the following properties:

Extremely compact

The average number of bits that are required to represent a node in a tree with n nodes depends on the average fanout (the average number of children of a node). For small fanouts, (0-7) the size is about 6*logAn bits, where A is the average fanout. A node in an organizational hierarchy of 100,000 people with an average fanout of 6 levels takes about 38 bits. This is rounded up to 40 bits, or 5 bytes, for storage.

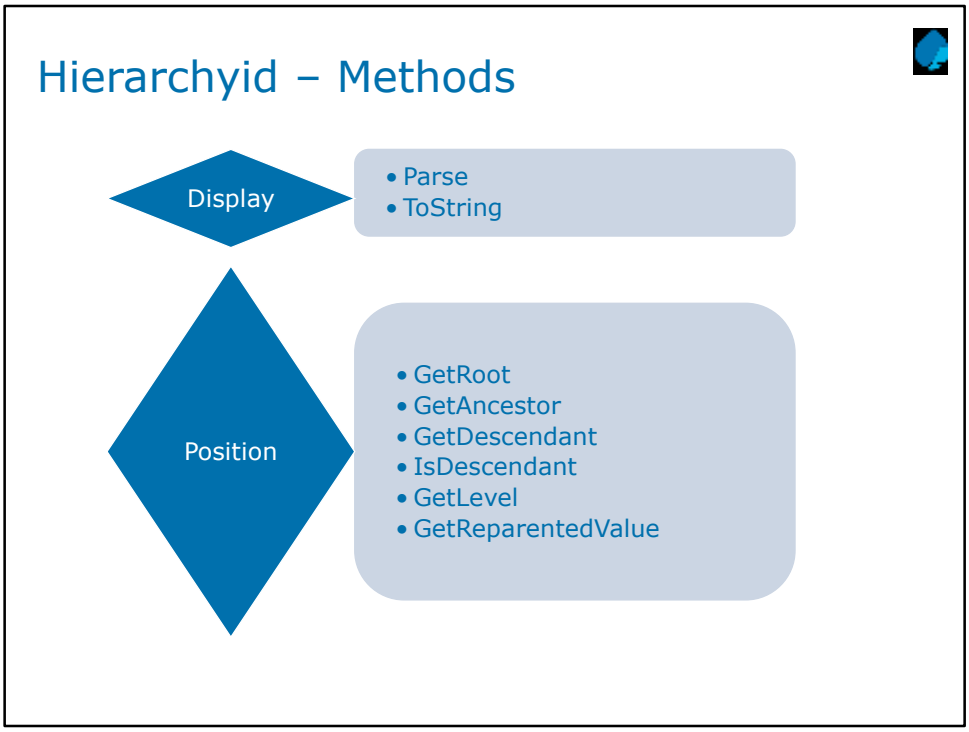
Comparison is in depth-first order

Given two hierarchyid values a and b, a<b means a comes before b in a depth-first traversal of the tree. Indexes on hierarchyid data types are in depth-first order, and nodes close to each other in a depth-first traversal are stored near each other. For example, the children of a record are stored adjacent to that record.

Support for arbitrary insertions and deletions

By using the GetDescendant method, it is always possible to generate a sibling to the right of any given node, to the left of any given node, or between any two siblings. The comparison property is maintained when an arbitrary number of nodes is inserted or deleted from the hierarchy. Most insertions and deletions preserve the compactness property. However, insertions between two nodes will produce hierarchyid values with a slightly less compact representation.

Instructor Notes:



Hierarchyid Methods

Parse: Converts a string representation of a hierarchy to a Hierarchyid value. Static.
ToString: Returns a string that contains the logical representation of this Hierarchyid .

GetRoot: Returns the root Hierarchyid node of this hierarchy tree. Static.
GetAncestor: Returns a Hierarchyid that represents the nth ancestor of this HierarchyID node.
GetDescendant: Returns a child node of this Hierarchyid node.
IsDescendant: Returns true if the passed-in child node is a descendant of this HierarchyID node.
GetLevel: Returns an integer that represents the depth of this Hierarchyid node in the overall hierarchy.
GetReparentedValue: Returns a node whose path from the root is the path to newRoot, followed by the path from oldRoot to this.

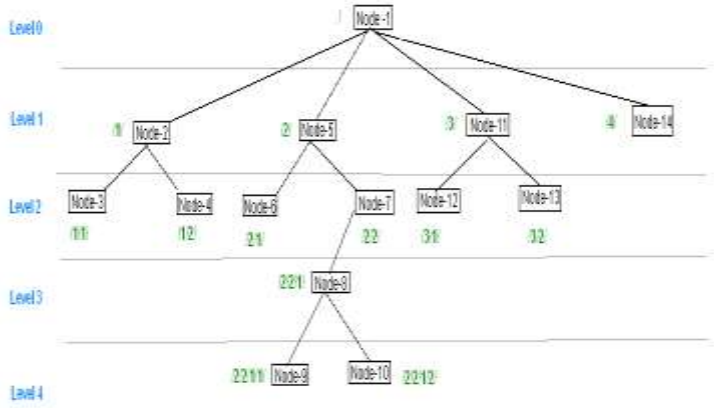
Instructor Notes:

Example:

```
-- Create database and table
CREATE DATABASE HierarchyDB

GO

USE HierarchyDB

GO

IF ( Object_id('HierarchyTab') > 0 )
DROP TABLE HierarchyTab

GO

CREATE TABLE HierarchyTab
(
NodeId     INT IDENTITY(1, 1)
,NodeDepth VARCHAR(100) NOT NULL
,NodePath  HIERARCHYID NOT NULL
,NodeDesc  VARCHAR(100)
)

GO

-- Creating constraint on hierarchy data type.
ALTER TABLE HierarchyTab ADD CONSTRAINT U_NodePath UNIQUE
   CLUSTERED (NodePath)

GO

-- Inserting data in above created table.
INSERT INTO HierarchyTab(NodeDepth,NodePath,NodeDesc)
VALUES
('1',HIERARCHYID::Parse('/'),'Node-1'),
('1.1',HIERARCHYID::Parse('/1/'),'Node-2'),
('1.1.1',HIERARCHYID::Parse('/1/1/'),'Node-3'),
('1.1.2',HIERARCHYID::Parse('/1/2/'),'Node-4'),
('1.2',HIERARCHYID::Parse('/2/'),'Node-5'),
('1.2.1',HIERARCHYID::Parse('/2/1/'),'Node-6'),
('1.2.2',HIERARCHYID::Parse('/2/2/'),'Node-7'),
('1.2.2.1',HIERARCHYID::Parse('/2/2/1/'),'Node-8'),
('1.2.2.1.1',HIERARCHYID::Parse('/2/2/1/1/'),'Node-9'),
('1.2.2.1.2',HIERARCHYID::Parse('/2/2/1/2/'),'Node-10'),
('1.3',HIERARCHYID::Parse('/3/'),'Node-11'),
('1.3.1',HIERARCHYID::Parse('/3/1/'),'Node-12'),
('1.3.2',HIERARCHYID::Parse('/3/2/'),'Node-13'),
('1.4',HIERARCHYID::Parse('/4/'),'Node-14')

GO
```

Instructor Notes:

Logical Representation of the table :



Working with Methods to display data from the table:

-- GetRoot()

```
SELECT
HIERARCHYID::GetRoot() AS RootNode,
HIERARCHYID::GetRoot().ToString() AS RootNodePath
GO
```

Result :

```
RootNode        RootNodePath
0x/
```

-- ToString()

```
SELECT
NodePath.ToString() AS NodeStringPath
,NodeId
,NodeDepth
,NodePath
,NodeDesc
FROM   HierarchyTab
GO
```

Instructor Notes:

Result :

NodeStringPath	NodeId	NodeDepth	NodePath	NodeDesc
/	1	1	0x	Node-1
/1/	2	1.1	0x58	Node-2
/1/1/	3	1.1.1	0x5AC0	Node-3
/1/2/	4	1.1.2	0x5B40	Node-4
/2/	5	1.2	0x68	Node-5
/2/1/	6	1.2.1	0x6AC0	Node-6
/2/2/	7	1.2.2	0x6B40	Node-7
/2/2/1/	8	1.2.2.1	0x6B56	Node-8
/2/2/1/1/	9	1.2.2.1.1	0x6B56B0	Node-9
/2/2/1/2/	10	1.2.2.1.2	0x6B56D0	Node-10
/3/	11	1.3	0x78	Node-11
/3/1/	12	1.3.1	0x7AC0	Node-12
/3/2/	13	1.3.2	0x7B40	Node-13
/4/	14	1.4	0x84	Node-14

Instructor Notes:

Introduction to Alias Data Types (User Defined Data Types)

- An alias data type is a user defined custom data type based on system-supplied data type
- In SQL Server alias data types offers a great deal of data consistency while working with various tables or databases
- You should create an alias data type when :
  - You need to define a commonly used data element with specific format
  - Database tables must store the same type of data in a column
  - These columns have identical data type, length & nullability
- Example

```
CREATE TYPE EmailAddress
FROM varchar(30) NOT NULL;
```

Introduction to Alias Data Types (User Defined Data Types) :

An alias data type is a user defined custom data type based on system supplied data type. An alias data type allows you to refine a data type further to guarantee
consistency when working with common data elements in various tables or databases.
It provides you with a convenient way to standardize the usage of native data types for columns that have same domain of possible values. Assume that you need to store many email addresses in your database, in various tables. As there is no single & definitive way to store Email Address, it is very hard to maintain consistency in all those tables which store email address in a column. However, if the Email Address will be used regularly throughout the database, you could define a EmailAddress alias data type and use that instead. This also makes it easier to understand the object definitions and code in your database.

1. Name
2. System data type upon which the new data type is based Nullability (whether the data type allows null values)

Note - When nullability is not explicitly defined, it will be assigned based on the ANSI null default setting for the database or connection.

Instructor Notes:

Dropping Alias Data Types

- You can drop alias data type by using Object Explorer in SQL Server Management Studio
- You can also use DROP TYPE Transact-SQL Statement to remove the alias data type from a database
- The DROP TYPE statement removes an alias data type from the current database
- You cannot drop a user-defined type until all references to that type have been removed
- Example

```
DROP TYPE EmailAddress
```

Note that when you create an alias data type, you can specify its nullability. An alias data type created with the NOT NULL option can never be used to store a NULL value.
It is important to specify the appropriate nullability when you create a data type, because it can be a lengthy procedure to change a data type. You must use the DROP TYPE statement to drop the data type and then re-create a new data type to replace it.
Because you cannot drop a data type that is used by tables in the database, you also need to ALTER every table that uses the data type first.

Tip – Alias data types that you create in the model database are automatically included in all databases that are subsequently created. However, if the data type is created in a user-defined database, the data type exists only in that user- defined database.

Dropping Alias Data Types:

You can drop alias data types by using Object Explorer in SQL Server Management studio or by using DROP TYPE Transact-SQL statement.
The DROP TYPE statement will not execute when :

1. There are tables in the database that contain columns of the alias data type or the user-defined type.
2. There are computed columns, CHECK constraints, schema-bound views, and schema-bound functions whose definitions reference the alias or user-defined type.
3. There are functions, stored procedures, or triggers created in the database, and these routines use variables and parameters of the alias or user-defined type.

Instructor Notes:

Demo

- Creating alias data types (UDTs)
- Dropping alias data types (UDTs)

Instructor Notes:

Data Types in SQL Server –Best Practices

- If column length varies, use a variable data type
- Use tinyint appropriately
- For numeric data types, commonly use decimal
- If storage is greater than 8000 bytes, use text or image
- Use money for currency
- Do not use float or real as primary keys
- If your character data type columns use the same or a similar number of characters consistently, use fixed length data types (char, nchar)
- Choose the smallest numeric or character data type required to store the data
- If you are going to be using a column for frequent sorts, consider an integer-based column rather than a character-based column

Instructor Notes:

What is a Database Schema?

- A Database Schema is a way to logically group SQL Server objects such as tables, views, stored procedures etc
- A schema is a distinct namespace, a container of SQL Server objects, distinct from users those who have created those objects
- In earlier releases of SQL Server, an object's namespace was determined by the user name of its owner
- An object owned by a database user is no longer tied to that user
- By introducing Schemas in database objects can now be manipulated independently of user
- A SQL Server user can be defined with default schema
- If no default schema is defined, sql server will assume dbo as the default schema

What is a Database Schema?

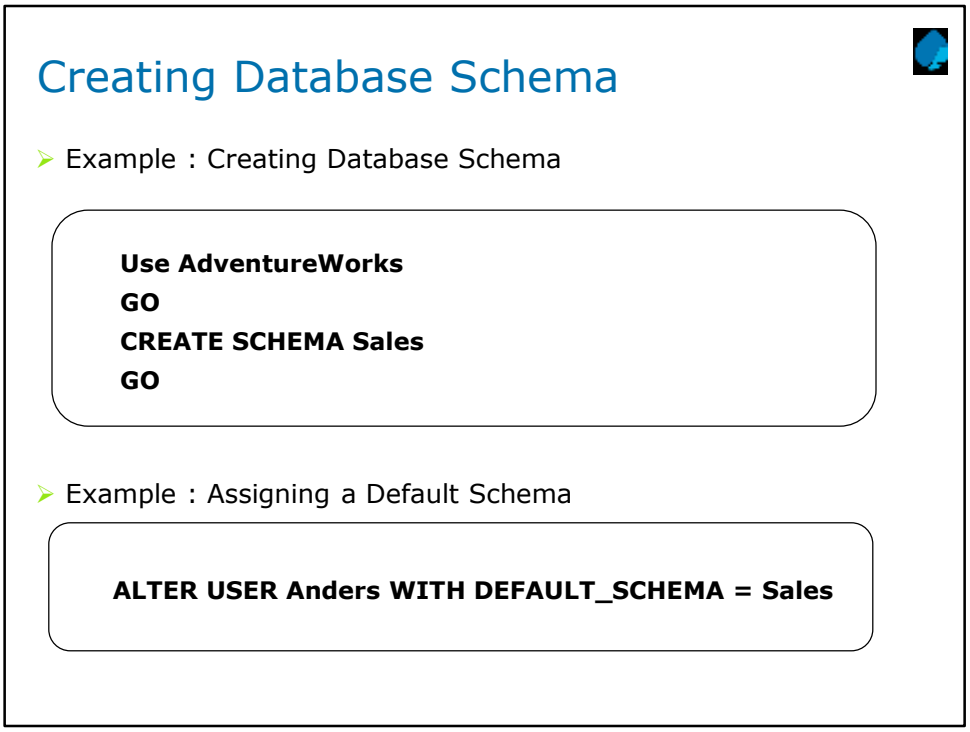
Microsoft introduced the concept of database schemas. A schema is an independent entity- a container of objects distinct from the user who created those objects. A schema is a distinct namespace to facilitate the separation, management, and ownership of database objects. It removed the tight coupling of database objects and owners.

In earlier releases of SQL Server, database object owners and users were the same things. This meant that if, say, a user creates a table in the database, that user cannot be deleted without deleting the table or first transferring it to another user. SQL Server introduced the concept of database schemas and the separation between database objects and ownership by users. An object owned by a database user is no longer tied to that user. The object now belongs to a schema – a container that can hold many database objects. The schema owner may own one or many schemas. This concept creates opportunities to expose database objects within a database for consumption yet protect them from modification, direct access using poor query techniques, or removal by users other than the owner.

When a schema is created, it is owned by a principal. A principal is any entity or object that has access to SQL Server resources. These are:

- Windows domain logins
- Windows local logins
- SQL Server logins
- Windows groups
- Database roles
- Server roles
- Application roles

Instructor Notes:

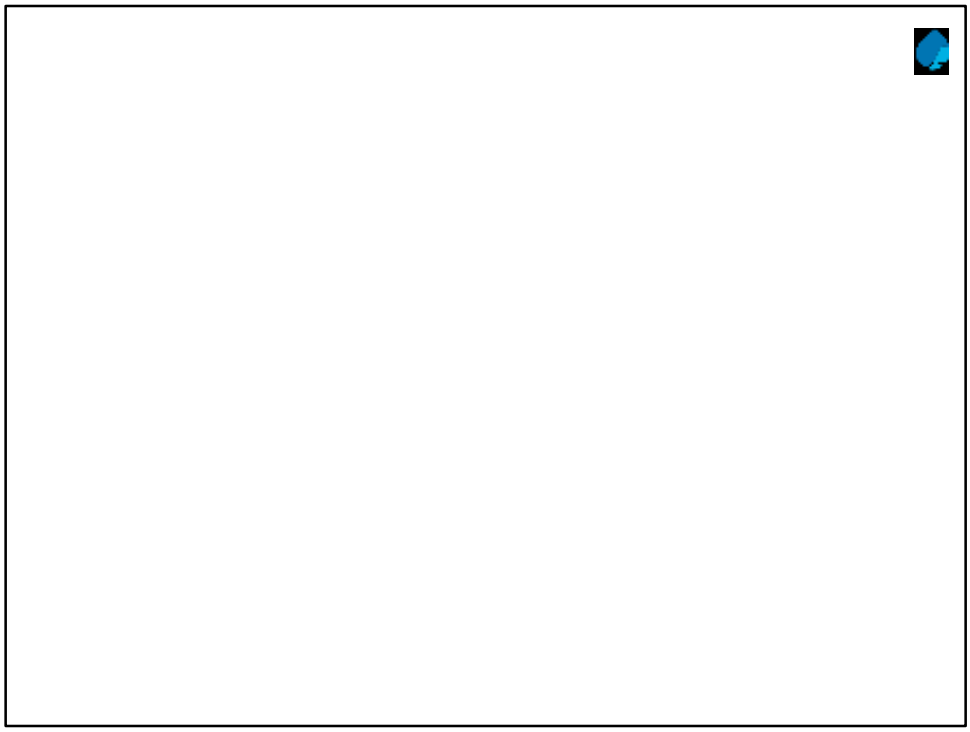


Creating Database Schema
As shown on the above slide, you can create a database schema by using CREATE SCHEMA Transact-SQL Statement.

The dbo Schema
Every database contains a schema named dbo. The dbo schema is the default schema for all users who do not have any other explicitly defined default schema.

How Object Name Resolution Works?
When a database contains multiple schemas, object name resolution can become confusing. For example, a database might contain two tables named Order in two different schemas, Sales and dbo. The qualified names of the objects within the
Database are unambiguous: Sales.Order and dbo.Order, respectively. However, the use of the unqualified name Order can produce unexpected results. You can assign users a default schema to control how unqualified object names are resolved.

Instructor Notes:



How Name Resolution Works?

SQL Server uses the following process to resolve an unqualified object name:

1. If the user has a default schema, SQL Server attempts to find the object in the default schema.
2. If the object is not found in the user's default schema, or if the user has no default schema, SQL Server attempts to find the object in the dbo schema.

For example, a user with the default schema Person executes the following Transact- SQL statement.

SELECT * FROM Contact

SQL Server will first attempt to resolve the object name to Person.Contact.
If the Person schema does not contain an object named Contact, SQL Server will attempt to resolve the object name to dbo.Contact.

If a user with no defined default schema executes the same statement, SQL Server will immediately resolve the object name to dbo.Contact.

Instructor Notes:

Demo

- Creating Database Schema

Instructor Notes:

Create Table

- Once you define all the data types in your database, you can create the Tables to store your business data
- The Table is the most important and central object of any RDBMS
- These tables may be temporary, lasting only for a single interactive SQL Session
- They also can be permanent, lasting weeks or for months
- Creating a database table involves :
  - Naming the table
  - Defining the columns
  - Assigning properties to the column
- In SQL Server, you can use CREATE TABLE Transact-SQL Statement for creating table

Create table in SQL Server
After you have designed the database , the tables that will store the data in the database can be created. The data is usually stored in permanent tables. Tables are stored in the database files until they are deleted and are available to any user who has the appropriate permissions.

Temporary Tables
You can also create temporary tables. Temporary tables are similar to permanent tables, except temporary tables are stored in tempdb and are deleted automatically when no longer in use.
The two types of temporary tables, local and global, differ from each other in their names, their visibility, and their availability. Local temporary tables have a single number sign (#) as the first character of their names; they are visible only to the current connection for the user; and they are deleted when the user disconnects from instances of SQL Server. Global temporary tables have two number signs (##) as the first characters of their names; they are visible to any user after they are created; and they are deleted when all users referencing the table disconnect from SQL Server.

Example
If you create a table named employees, the table can be used by any person who has the security permissions in the database to use it, until the table is deleted. If you create a local temporary table named #employees, you are the only person who can work with the table, and it is deleted when you disconnect.

```
CREATE TABLE #Employee
(
                EmpID           INT,
                Name            VARCHAR(20),
                DOJ             DATETIME
)
```

Instructor Notes:

If you create a global temporary table named ##employees, any user in the database can work with this table. If no other user works with this table after you create it, the table is deleted when you disconnect. If another user works with the table after you create it, SQL Server deletes it when both of you disconnect.

Table Properties

You can define up to 1,024 columns per table. Table and column names must follow the rules for identifiers; they must be unique within a given table, but you can use the same column name in different tables in the same database. You must also define a data type for each column.

Although table names must be unique for each owner within a database, you can create multiple tables with the same name if you specify different owners for each. You can create two tables named employees and designate Jonah as the owner of one and Sally as the owner of the other. When you need to work with one of the employees tables, you can distinguish between the two tables by specifying the owner with the name of the table.

Tables & Columns - Naming Guidelines

1. Use Pascal Casing (also known as upper camel casing)
2. Avoid abbreviations
3. A long name that users understand is preferred over a short name that users might not understand

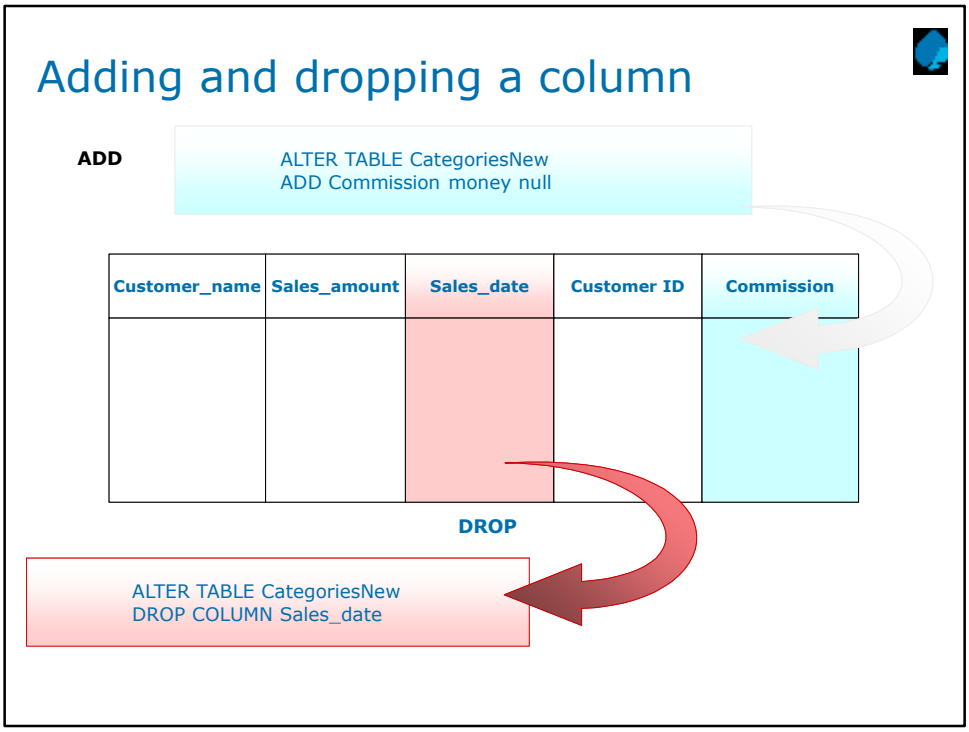
Creating Table

- Given a Table Structure for Employee

Column Name	Data Type	Nullability
Employee_Code	Int	NOT NULL
Employee_Name	Varchar(40)	NOT NULL
Employee_DOB	Datetime	NOT NULL
Employee_EmailID	Varchar(20)	NULL

```
CREATE TABLE Employee
(
    Employee_Code     int          NOT NULL,
    Employee_Name     varchar(40)  NOT NULL,
    Employee_DOB      datetime     NOT NULL,
    Employee_EmailID  varchar(20)  NULL
)
```

Instructor Notes:



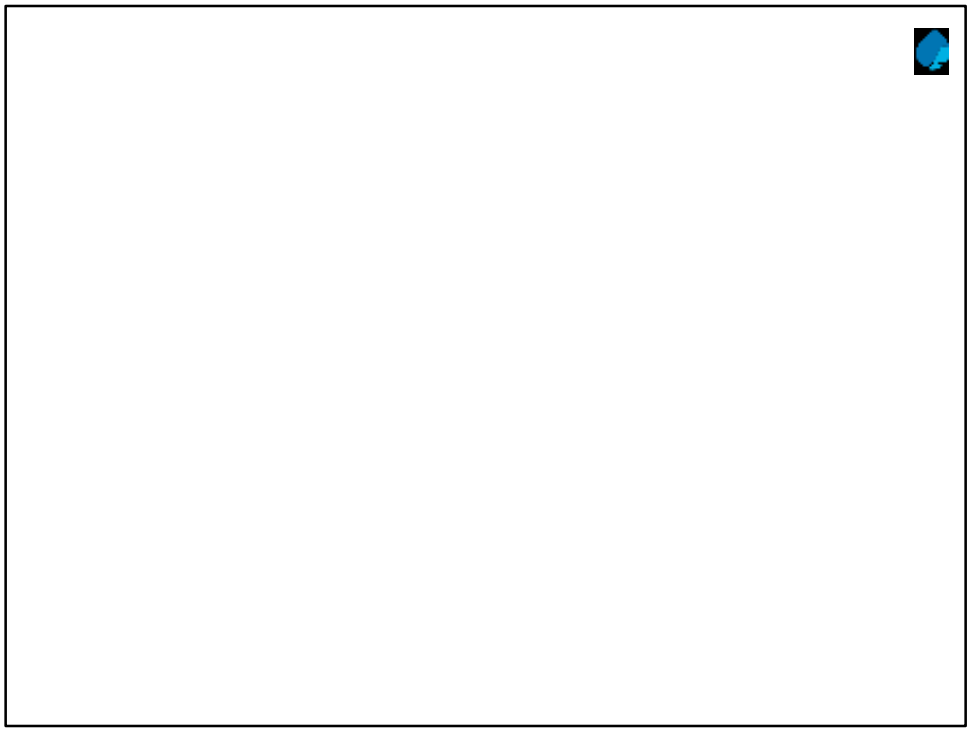
Modifying tables in SQL Server

After a table is created, you can change many of the options that were defined for the table when it was originally created. You can modify tables in many different ways.
You can modify tables to make changes to the columns, constraints, and indexes associated with tables. You can use ALTER TABLE statement to implement most common modifications to table

Using ALTER TABLE Columns can be added, modified, or deleted. For example, the column name, length, data type, precision, scale, and nullability can all be changed, although some restrictions exist. PRIMARY KEY and FOREIGN KEY constraints can be added or deleted. UNIQUE and CHECK constraints and DEFAULT definitions (and objects) can be added or deleted.

Adding & Dropping Columns :
When you use the ALTER TABLE statement to add a column, the new column is added at the end of the table.
There are also some things you need to consider with regard to the null option specified for a new column. In the case of a column that allows nulls, there is no real issue. SQL Server adds the column and allows a NULL value for all rows. If NOT NULL is specified, however, the column must be an identity column or have a default specified. Note that even if a default is specified, if the column allows nulls, the column does not be populated with the default. You use the WITH VALUES clause as part of the default specification to override this and populate the column with the default.

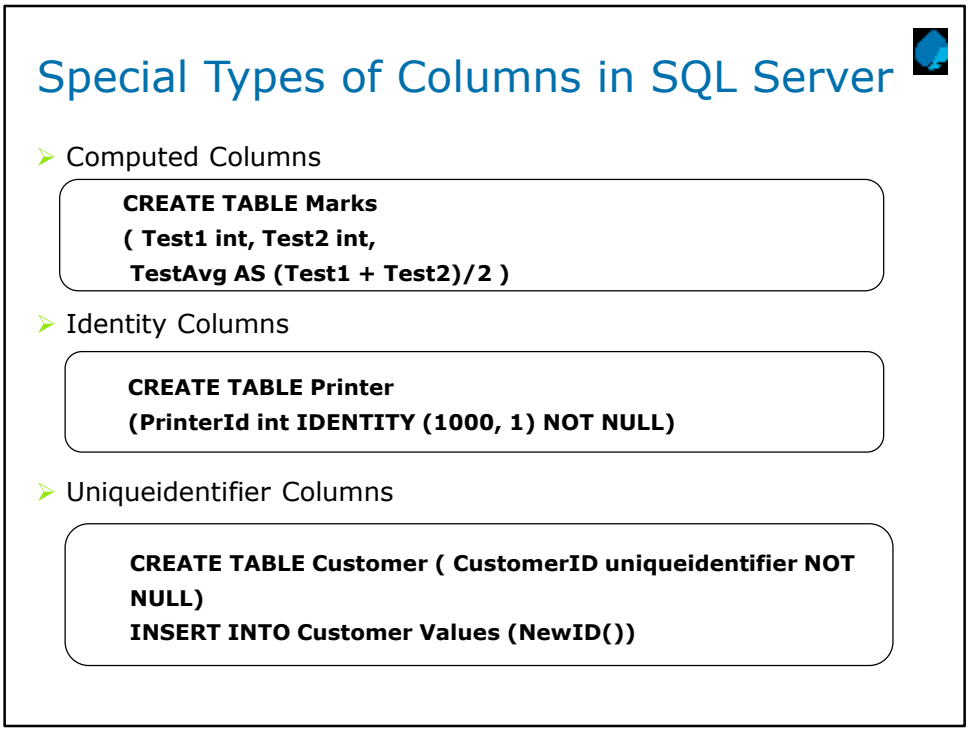
Instructor Notes:



With some restrictions, columns can also be dropped from a table.
The following columns cannot be dropped:

- A column in a schema-bound view
- An indexed column
- A replicated column
- A column used in a CHECK, FOREIGN KEY, UNIQUE, or PRIMARY KEY constraints
- A column that is associated with a default or bound to a default object
- A column that is bound to a rule

Instructor Notes:



Special Types of Columns in SQL Server

Computed Columns - A computed column is a column whose value is calculated based on other columns. Generally speaking, the column is a virtual column because it is calculated on-the-fly, and no value is stored in the database table. With SQL Server 2008, you have an option of actually storing the calculated value in the database. You do so by marking the column as persisted. If the computed column is persisted, you can create an index on this column as well.

Identity Columns - You can use the Identity property to create columns that contain system-generated sequential values that identify each row inserted into a table. Having SQL Server automatically provide key values can reduce costs and improve performance. It can be assigned only to columns that are of the following types:

decimal
int
numeric
smallint
bigint
Tinyint

Only one identity column can exist for each table, and that column cannot allow nulls. When implementing the IDENTITY property, you supply a seed and an increment. The seed is the starting value for the numeric count, and the increment is the amount by which it grows. A seed of 10 and an increment of 10 would produce values of 10, 20, 30, 40, and so on. If not specified, the default seed value is 1, and the increment is 1

Uniqueidentifier Columns - Columns defined with the uniqueidentifier data type can be used to store globally unique identifiers (GUIDs), which are guaranteed to be universally unique. You can generate a value for a uniqueidentifier column by using the NEWID Transact-SQL function.

Instructor Notes:

SEQUENCE

- SQL Server 2012 introduces Sequence as database object.
- Sequence is an object in each database and is similar to IDENTITY in its functionality.
- It can have start value, incrementing value and an end value defined in it.
- It can be added to a column whenever required rather than defining an identity column individually for tables.
- Limitations of using the Identity property can be overcome by the introduction of this new object SEQUENCE.

Instructor Notes:

SEQUENCE

- CREATESEQUENCE SQ1 AS INT
- START WITH 100
- INCREMENT BY 25
- MINVALUE 100
- MAXVALUE 200
- CYCLE
- CACHE 10
- GO
- SELECT NEXT VALUE FOR SQ1

CREATE SEQUENCE [schema_name .] sequence_name [AS [built_in_integer_type | user-defined_integer_type]] [START WITH <constant>] [INCREMENT BY <constant>] [{ MINVALUE [<constant>] } | { NOMINVALUE }] [{ MAXVALUE [<constant>] } | { NOMAXVALUE }] [CYCLE | { NOCYCLE }] [{ CACHE [<constant>] } | { NO CACHE }] [;]

sequence_name Specifies the unique name by which the sequence is known in the database. Type is **sysname**.
[built_in_integer_type | user-defined_integer_type] A sequence can be defined as any integer type. The following types are allowed.
tinyint - Range 0 to 255
smallint - Range -32,768 to 32,767
int - Range -2,147,483,648 to 2,147,483,647
bigint - Range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
decimal and **numeric** with a scale of 0.
Any user-defined data type (alias type) that is based on one of the allowed types.
If no data type is provided, the **bigint** data type is used as the default.
START WITH <constant>The first value returned by the sequence object. The **START** value must be a value less than or equal to the maximum and greater than or equal to the minimum value of the sequence object. The default start value for a new sequence object is the minimum value for an ascending sequence object and the maximum value for a descending sequence object.
INCREMENT BY <constant>Value used to increment (or decrement if negative) the value of the sequence object for each call to the **NEXT VALUE FOR** function. If the increment is a negative value, the sequence object is

Instru

descending; otherwise, it is ascending. The increment cannot be 0. The default increment for a new sequence object is 1.

Instructor Notes:

SEQUENCE

```
CREATE SEQUENCE mySeq
START WITH 10
INCREMENT BY 5;

SELECT NEXT VALUE FOR mySeq;
SELECT NEXT VALUE FOR mySeq;
SELECT NEXT VALUE FOR mySeq;

create table myTable
(sid int, Sname varchar(15))

INSERT INTO myTable(sid, sname)
VALUES (NEXT VALUE FOR mySeq, 'Tom’);

select * from myTable

INSERT INTO myTable(sid, sname)
VALUES (NEXT VALUE FOR mySeq, 'Moody');
```

[MINVALUE <constant> | NO MINVALUE]Specifies the bounds for the sequence object. The default minimum value for a new sequence object is the minimum value of the data type of the sequence object. This is zero for the **tinyint** data type and a negative number for all other data types.
[MAXVALUE <constant> | NO MAXVALUE Specifies the bounds for the sequence object. The default maximum value for a new sequence object is the maximum value of the data type of the sequence object.
[CYCLE | NO CYCLE]Property that specifies whether the sequence object should restart from the minimum value (or maximum for descending sequence objects) or throw an exception when its minimum or maximum value is exceeded. The default cycle option for new sequence objects is NO CYCLE.
Note that cycling restarts from the minimum or maximum value, not from the start value.
[CACHE [<constant>] | NO CACHE]Increases performance for applications that use sequence objects by minimizing the number of disk IOs that are required to generate sequence numbers. Defaults to CACHE.
For example, if a cache size of 50 is chosen, SQL Server does not keep 50 individual values cached. It only caches the current value and the number of values left in the cache. This means that the amount of memory required to store the cache is always two instances of the data type of the sequence object.

Instructor Notes:

Demo

- Creating Table

Instructor Notes:

Enforcing Data Integrity

- The term Data Integrity refers to correctness and completeness of the data in a database
- To preserve the consistency and accuracy of data, every RDBMS imposes one or more data integrity constraints in a database
- These constraints restricts the wrong or invalid data values that can be inserted or updated in a database
- How and what kind of integrity is enforced in the database depends on the type integrity being enforced
- Enforcing data integrity ensures quality of the data in the database
- The quality of data refers to :
  - Accuracy
  - Completeness
  - Consistent
  - Correct

An important step in database planning is deciding the best way to enforce the integrity of the data. Data integrity refers to the consistency and accuracy of data that is stored in a database.

What Is Data Integrity?

The term *data integrity refers to the correctness and completeness of the data in a* database. When the contents of a database are modified with the INSERT, DELETE, or UPDATE statements, the integrity of the stored data can be lost in many different ways.

For example:

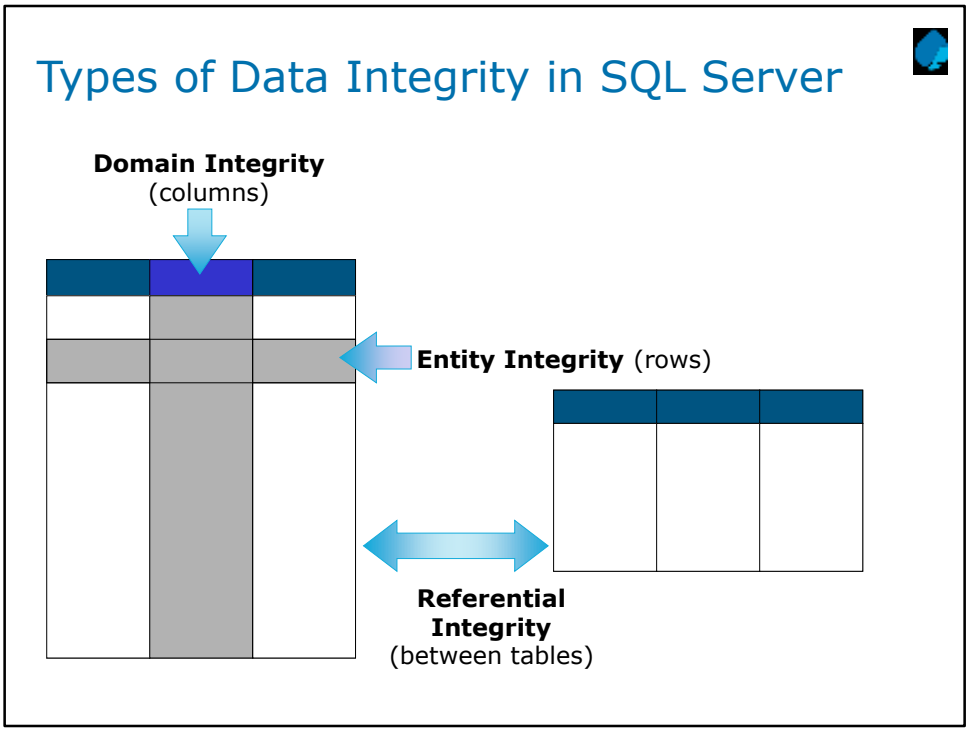
- Invalid data may be added to the database, such as an Employee that belongs to a nonexistent department.
- Existing data may be modified to an incorrect value, such as reassigning an Employee to a nonexistent project.
- Changes to the database may be lost due to a system error or power failure.
- Changes may be partially applied, such as adding an order for a product without adjusting the quantity available for sale.

One of the important roles of a relational DBMS is to preserve the integrity of its stored data to the greatest extent possible.

Enforcing data integrity ensures the quality of the data in the database.

Instructor Notes:

To ensure data integrity in a table we need enforce data some constraints.
Explain with examples
Domain, entity and referential integrity



Types of Data Integrity in SQL Server

Enforcing data integrity guarantees the quality of the data in the database. For example, if an employee is entered with an employee ID value of 123, the database should not permit another employee to have an ID with the same value. If you have an employee_rating column intended to have values ranging from 1 to 5, the database should not accept a value outside that range. If the table has a dept_id column that stores the department number for the employee, the database should permit only values that are valid for the department numbers in the company.
Two important steps in planning tables are to identify valid values for a column and to decide how to enforce the integrity of the data in the column. Data integrity falls into the following categories:
Entity integrity
Domain integrity
Referential integrity

Instructor Notes:

There are two ways of defining data integrity
1. at object level
2. by writing a script

Ways of implementing Data Integrity in SQL Server

- Declarative Data Integrity
  - Criteria defined in object definitions
  - SQL Server enforces automatically
  - Implement by using constraints, defaults, and rules
  - Simplest integrity check
- Procedural Data Integrity
  - Implement by using triggers, stored procedures & application code
  - Data validation criteria will be defined in script
  - Allows more advanced data integrity checks

There are two ways to implement data integrity in SQL Server, either using declarative data integrity or procedural data integrity.

Declarative Data Integrity – It is a set of rules that are applied to a table and its columns using the CREATE TABLE or ALTER TABLE statements. These rules are called constraints, rules and defaults. This is the preferred and simplest method of enforcing integrity because it has low overhead and requires little or no custom programming.

Procedural Data Integrity - Procedural integrity can be implemented with stored procedures, triggers, and application code. It allows more advanced integrity check as You can implement the custom code in many different ways to enforce the integrity of your data.

Instructor Notes:

Giving example for each type of constraint would help.

Determining the type of constraint to be used

Type of integrity	Constraint type
Domain	DEFAULT
	CHECK
	REFERENTIAL
Entity	PRIMARY KEY
	UNIQUE
Referential	FOREIGN KEY
	CHECK

Determining the type of constraint to be used

Entity Integrity
Entity integrity defines a row as a unique entity for a particular table. Entity integrity enforces the integrity of the identifier columns or the primary key of a table, through UNIQUE indexes, UNIQUE constraints or PRIMARY KEY constraints.

Domain Integrity
Domain integrity is the validity of entries for a specific column. You can enforce domain integrity to restrict the type by using data types, restrict the format by using CHECK constraints and rules, or restrict the range of possible values by using FOREIGN KEY constraints, CHECK constraints, DEFAULT definitions, NOT NULL definitions, and rules.

Referential Integrity
Referential integrity preserves the defined relationships between tables when rows are entered or deleted. In SQL Server, referential integrity is based on relationships between foreign keys and primary keys or between foreign keys and unique keys, through FOREIGN KEY and CHECK constraints. Referential integrity makes sure that key values are consistent across tables. This kind of consistency requires that there are no references to nonexistent values and that if a key value changes, all references to it change consistently throughout the database.
When you enforce referential integrity, SQL Server prevents users from doing the following:

- Adding or changing rows to a related table if there is no associated row in the primary table.
- Changing values in a primary table that causes orphaned rows in a related table.
- Deleting rows from a primary table if there are matching related rows.

Instructor Notes:

We need to clearly tell them about table level and column constraint.

Creating Constraints

- Use CREATE TABLE or ALTER TABLE
- Can Add Constraints to a Table with existing Data
- Can Place Constraints on Single or Multiple Columns
  - Single column, called column-level constraint
  - Multiple columns, called table-level constraint

You must declare constraints that operate on more than one column as table-level constraints.
For example, the following create table statement has a check constraint that operates on two columns, pub_id and pub_name:

```
create table my_publishers
(
pub_id char(4), pub_name varchar(40),
constraint my_chk_constraint check (pub_id in ("1234", “4321", “1212") or pub_name not like "Bad Books")
)
```

You can declare constraints that operate on just one column as column-level constraints, but it is not required. For example, if the above check constraint uses only one column (pub_id), you can place the constraint on that column:

```
create table my_publishers
(
pub_id char(4) constraint my_chk_constraint check (pub_id in ("1389", "0736", "0877")),
pub_name varchar(40)
)
```

In either case, the constraint keyword and accompanying constraint_name are optional.

Instructor Notes:

Consideration for using constraints

- Can be changed without recreating a table
- Require error-checking in applications and transactions
- Verify existing data

Instructor Notes:

Types constraints

- DEFAULT Constraints
- CHECK Constraints
- PRIMARY KEY Constraints
- UNIQUE Constraints
- FOREIGN KEY Constraints
- Cascading Referential Integrity

SQL Server supports the following classes of constraints:
NOT NULL specifies that the column does not accept NULL values.

CHECK constraints enforce domain integrity by limiting the values that can be put in a column. A CHECK constraint specifies a Boolean (evaluates to TRUE, FALSE, or unknown) search condition that is applied to all values that are entered for the column. All values that evaluate to FALSE are rejected. You can specify multiple CHECK constraints for each column.

UNIQUE constraints enforce the uniqueness of the values in a set of columns. In a UNIQUE constraint, no two rows in the table can have the same value for the columns. Primary keys also enforce uniqueness, but primary keys do not allow for NULL as one of the unique values.

PRIMARY KEY constraints identify the column or set of columns that have values that uniquely identify a row in a table.
No two rows in a table can have the same primary key value. You cannot enter NULL for any column in a primary key. We recommend using a small, integer column as a primary key. Each table should have a primary key. A column or combination of columns that qualify as a primary key value is referred to as a candidate key.

FOREIGN KEY constraints identify and enforce the relationships between tables.

Instructor Notes:

DEFAULT constraints

- Apply only to INSERT statements
- Only one DEFAULT constraint per column
- Cannot be used with IDENTITY property or rowversion data type
- Allow some system-supplied values

```
USE Northwind
ALTER TABLE dbo.Customers
ADD
CONSTRAINT DF_contactname DEFAULT 'UNKNOWN'
FOR ContactName
```

Defaults specify what values are used in a column if you do not specify a value for the column when you insert a row. Defaults can be anything that evaluates to a constant, such as a constant, built-in function, or mathematical expression.

To apply defaults, create a default definition by using the DEFAULT keyword in CREATE TABLE. This assigns a constant expression as a default on a column.

Each column in a record must contain a value, even if that value is NULL. There may be situations when you must load a row of data into a table but you do not know the value for a column, or the value does not yet exist. If the column allows for null values, you can load the row with a null value. Because nullable columns may not be desirable, a better solution could be to define, where appropriate, a DEFAULT definition for the column. For example, it is common to specify zero as the default for numeric columns, or N/A as the default for string columns when no value is specified.

When you load a row into a table with a DEFAULT definition for a column, you implicitly instruct the SQL Server Database Engine to insert a default value in the column when a value is not specified for it.

Note: You can also use the DEFAULT VALUES clause of the INSERT STATEMENT to explicitly instruct the Database Engine to insert a default value for a column.

If a column does not allow for null values and does not have a DEFAULT definition, you must explicitly specify a value for the column, or the Database Engine returns an error that states that the column does not allow null values.

The value inserted into a column that is defined by the combination of the DEFAULT definition and the nullability of the column can be summarized as shown in the following table.

Column definition	**No entry, no DEFAULT**	**No entry, DEFAULT**	**Enter null value**
Allows null value	NULL	Default value	NULL
Disallows null values	Error	Default value	Error

Instructor Notes:

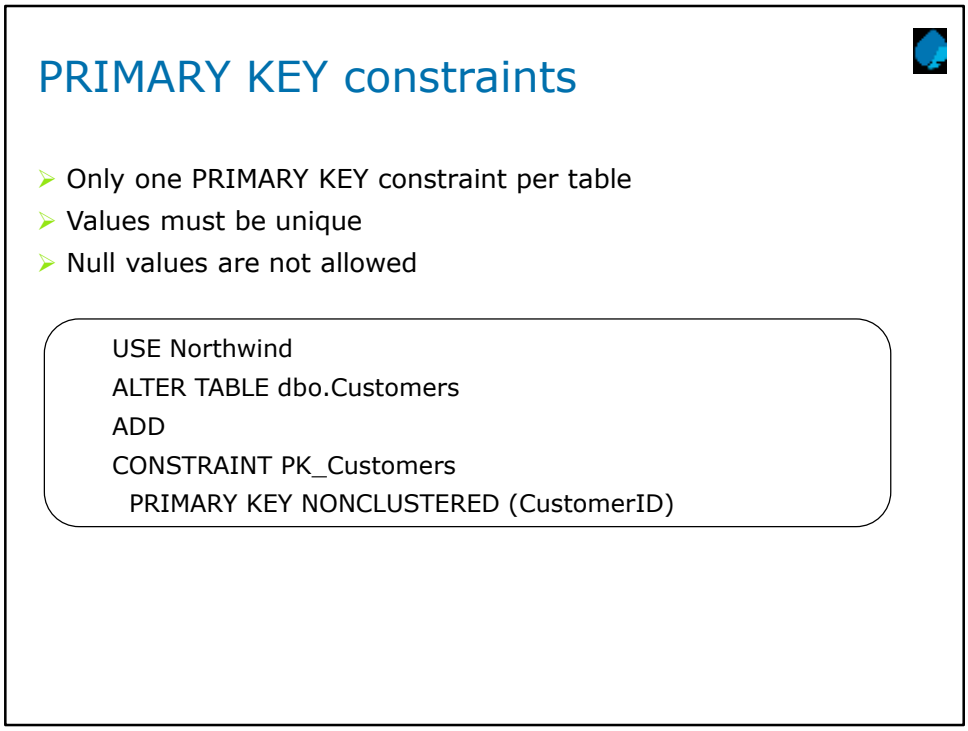
CHECK constraints

- Are used with INSERT and UPDATE statements
- Can reference other columns in the same table
- Cannot:
  - Be used with the rowversion data type
  - Contain subqueries

```
USE Northwind
ALTER TABLE dbo.Employees
ADD
CONSTRAINT CK_birthdate
CHECK (BirthDate > '01-01-1900' AND BirthDate < getdate())
```

You can declare a **check** constraint to limit the values users insert into a column in a table. Check constraints are useful for applications that check a limited, specific range of values. A **check** constraint specifies a *search_condition* that any value must pass before it is inserted into the table. A *search_condition* can include:
A list of constant expressions introduced with **in**
A range of constant expressions introduced with **between**
A set of conditions introduced with **like**, which may contain wildcard characters
An expression can include arithmetic operations and Transact-SQL built-in functions. The *search_condition* cannot contain subqueries, a set function specification, or a target specification.

Instructor Notes:



A table typically has a column or combination of columns that contain values that uniquely identify each row in the table. This column, or columns, is called the primary key (PK) of the table and enforces the entity integrity of the table. You can create a primary key by defining a PRIMARY KEY constraint when you create or modify a table.

A table can have only one PRIMARY KEY constraint, and a column that participates in the PRIMARY KEY constraint cannot accept null values. Because PRIMARY KEY constraints guarantee unique data, they are frequently defined on an identity column.

When you specify a PRIMARY KEY constraint for a table, the SQL Server Database Engine enforces data uniqueness by creating a unique index for the primary key columns. This index also permits fast access to data when the primary key is used in queries. Therefore, the primary keys that are chosen must follow the rules for creating unique indexes.

If a PRIMARY KEY constraint is defined on more than one column, values may be duplicated within one column, but each combination of values from all the columns in the PRIMARY KEY constraint definition must be unique.

Instructor Notes:

UNIQUE constraints

- Allow One Null Value
- Allow Multiple UNIQUE Constraints on a Table
- Defined with One or More Columns

```
USE Northwind
ALTER TABLE dbo.Suppliers
ADD
CONSTRAINT U_CompanyName
  UNIQUE NONCLUSTERED (CompanyName)
```

You can use UNIQUE constraints to make sure that no duplicate values are entered in specific columns that do not participate in a primary key. Although both a UNIQUE constraint and a PRIMARY KEY constraint enforce uniqueness, use a UNIQUE constraint instead of a PRIMARY KEY constraint when you want to enforce the uniqueness of a column, or combination of columns, that is not the primary key.
Multiple UNIQUE constraints can be defined on a table, whereas only one PRIMARY KEY constraint can be defined on a table.
Also, unlike PRIMARY KEY constraints, UNIQUE constraints allow for the value NULL. However, as with any value participating in a UNIQUE constraint, only one null value is allowed per column.
A UNIQUE constraint can be referenced by a FOREIGN KEY constraint.

Instructor Notes:

FOREIGN KEY constraints

- Must Reference a PRIMARY KEY or UNIQUE Constraint
- Provide Single or Multicolumn Referential Integrity
- Do Not Automatically Create Indexes
- Users Must Have SELECT or REFERENCES Permissions on Referenced Tables
- Use Only REFERENCES Clause Within Same Table

```
USE Northwind
ALTER TABLE dbo.Orders
ADD CONSTRAINT FK_Orders_Customers
  FOREIGN KEY (CustomerID)
  REFERENCES dbo.Customers(CustomerID)
```

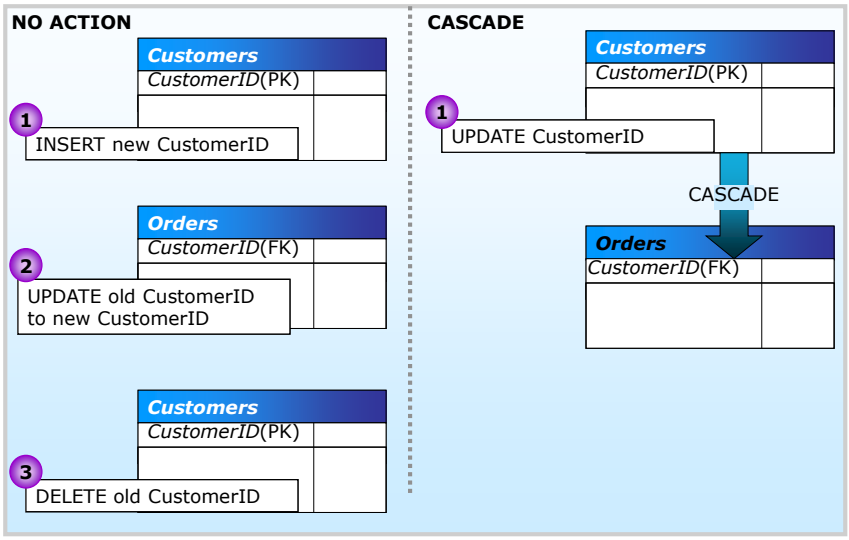
A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables. You can create a foreign key by defining a FOREIGN KEY constraint when you create or modify a table.

In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table.

A FOREIGN KEY constraint does not have to be linked only to a PRIMARY KEY constraint in another table; it can also be defined to reference the columns of a UNIQUE constraint in another table. A FOREIGN KEY constraint can contain null values; however, if any column of a composite FOREIGN KEY constraint contains null values, verification of all values that make up the FOREIGN KEY constraint is skipped. To make sure that all values of a composite FOREIGN KEY constraint are verified, specify NOT NULL on all the participating columns.

Instructor Notes:

Cascading Referential Integrity Constraints



Cascading Referential Integrity Constraints
Cascading referential integrity constraints allow you to define the actions SQL Server takes when a user attempts to delete or update a key to which existing foreign keys point.
The REFERENCES clauses of the CREATE TABLE and ALTER TABLE statements support ON DELETE and ON UPDATE clauses:

[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]

NO ACTION is the default if ON DELETE or ON UPDATE is not specified. NO ACTION specifies the same behavior that occurs in earlier versions of SQL Server.
CASCADE allows deletions or updates of key values to cascade through the tables defined to have foreign key relationships that can be traced back to the table on which the modification is performed. CASCADE cannot be specified for any foreign keys or primary keys that have a timestamp column.
ON DELETE CASCADE
Specifies that if an attempt is made to delete a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the rows deleted from those tables.
ON UPDATE CASCADE
Specifies that if an attempt is made to update a key value in a row, where the key value is referenced by foreign keys in existing rows in other tables, all of the foreign key values are also updated to the new value specified for the key. If cascading referential actions have also been defined on the target tables, the specified cascading actions are also taken for the key values updated in those tables.

Instructor Notes:

Demo

- Demonstration on all types of SQL Server Constraints

Instructor Notes:

Disabling Constraints

- Disabling constraint checking on existing data
- Disabling constraint checking when loading new data

Sometimes you need to perform some actions that require the FOREIGN KEY or CHECK constraints be disabled, for example, your company do not hire foreign employees, you made the appropriate constraint, but the situation was changed and your boss need to hire the foreign employee, but only this one. In this case, you need to disable the constraint by using the ALTER TABLE statement. After these actions will be performed, you can re-enable the FOREIGN KEY and CHECK constraints by using the ALTER TABLE statement.

Instructor Notes:

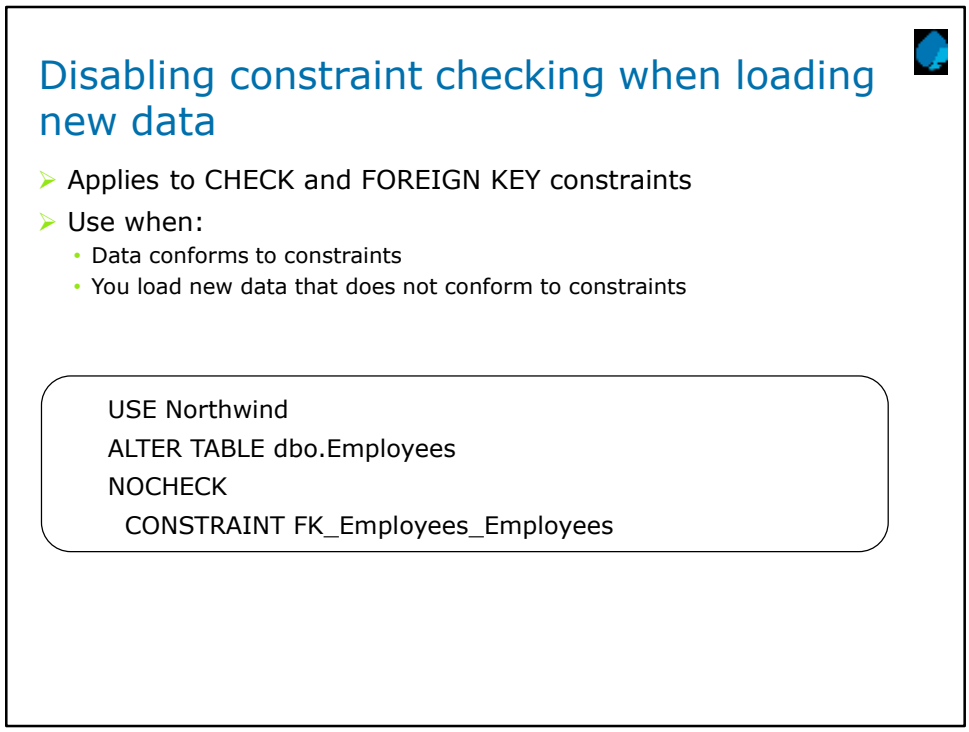
Disabling Constraint Checking on Existing Data

- Applies to CHECK and FOREIGN KEY constraints
- Use WITH NOCHECK option when adding a new constraint
- Use if existing data will not change
- Can change existing data before adding constraints

```
USE Northwind
ALTER TABLE dbo.Employees
WITH NOCHECK
  ADD CONSTRAINT FK_Employees_Employees
  FOREIGN KEY (ReportsTo)
  REFERENCES dbo.Employees(EmployeeID)
```

WITH CHECK | WITH NOCHECK
Specifies whether the data in the table is or is not validated against a newly added or re-enabled FOREIGN KEY or CHECK constraint. If not specified, WITH CHECK is assumed for new constraints, and WITH NOCHECK is assumed for re-enabled constraints.
If you do not want to verify new CHECK or FOREIGN KEY constraints against existing data, use WITH NOCHECK. We do not recommend doing this, except in rare cases. The new constraint will be evaluated in all later data updates. Any constraint violations that are suppressed by WITH NOCHECK when the constraint is added may cause future updates to fail if they update rows with data that does not comply with the constraint.
The query optimizer does not consider constraints that are defined WITH NOCHECK. Such constraints are ignored until they are re-enabled by using ALTER TABLE table CHECK CONSTRAINT ALL.

Instructor Notes:



Instructor Notes:

Using DEFAULTS & RULES

- As independent objects they:
  - Are defined once
  - Can be bound to one or more columns or user-defined data types

```
CREATE DEFAULT phone_no_default
  AS '(000)000-0000'
GO
EXEC sp_bindefault phone_no_default,
  'Customers.Phone'
```

```
CREATE RULE regioncode_rule
  AS @regioncode IN ('IA', 'IL', 'KS', 'MO')
GO
EXEC sp_bindrule regioncode_rule,
  'Customers.Region'
```

Using Defaults and Rules

Default :-

When bound to a column or an alias data type, a default specifies a value to be inserted into the column to which the object is bound (or into all columns, in the case of an alias data type), when no value is explicitly supplied during an insert.

CREATE DEFAULT [schema_name .] default_name
AS constant_expression [;]

A default name can be created only in the current database. Within a database, default names must be unique by schema. When a default is created, use sp_bindefault to bind it to a column or to an alias data type.

Rules : -

Creates an object called a rule. When bound to a column or an alias data type, a rule specifies the acceptable values that can be inserted into that column.

A column or alias data type can have only one rule bound to it. However, a column can have both a rule and one or more check constraints associated with it. When this is true, all restrictions are evaluated.

A rule can be created only in the current database. After you create a rule, execute sp_bindrule to bind the rule to a column or to alias data type. A rule must be compatible with the column data type. For example, "@value LIKE A%" cannot be used as a rule for a numeric column. A rule cannot be bound to a text, ntext, image, varchar(max), nvarchar(max), varbinary(max), xml, CLR user-defined type, or timestamp column. A rule cannot be bound to a computed column.

Instructor Notes:

Demo

- Disabling Constraints
- Working with DEFAULTS & RULES

Instructor Notes:

Deciding - Enforcement method to use

Data integrity components	Functionality	Performance costs	Before or after modification
Constraints	Medium	Low	Before
Defaults and rules	Low	Low	Before
Triggers	High	Medium-High	After
Data types, Null/Not Null	Low	Low	Before

Instructor Notes:

TRUNCATE

- Removes all rows from a table
- TRUNCATE TABLE is similar to the DELETE statement with no WHERE clause

```
TRUNCATE TABLE  Employees
```

Instructor Notes:

Dropping table

- At times you need to delete a table, for example when you want to implement a new design or free up space in the database
- You can use DROP TABLE Transact-SQL statement to drop the table from Database

```
DROP TABLE   Employee
```

- You can reference multiple tables in a single DROP TABLE command by separating the table names with comma

Dropping table SQL Server
At times you need to delete a table (for example, when you want to implement a new design or free up space in the database). When you delete a table, its structural definition, data, full-text indexes, constraints, and indexes are permanently deleted from the database, and the space formerly used to store the table and its indexes is made available for other tables. You can explicitly drop a temporary table if you do not want to wait until it is dropped automatically.

A big consideration when dropping a table is the table's relationship to other tables. If a foreign key references the table that you want to drop, the referencing table or foreign key constraint must be dropped first. In a database that has many related tables, this can get complicated. Fortunately, a few tools can help you through this. The system Stored procedure sp_helpconstraint is one of the tools. This procedure lists all the foreign key constraints that reference a table.

The other approach is to right-click the table in Object Explorer and choose View Dependencies. The dialog that appears gives you the option of viewing the objects that depend on the table or viewing the objects on which the table depends.

To delete a table - DROP TABLE
A. Drop a table in the current database
DROP TABLE titles
This example removes the titles table and its data and indexes from the current database.
B. Drop a table in another database
DROP TABLE pubs.dbo.authors
This example drops the authors table in the pubs database. It can be executed from any database.

Instructor Notes:

Commonly used T-SQL Statement

- Transact-SQL statements can be used for:
  - Creating a database
  - Creating and altering table
  - Insert, update & delete data in the table
  - Reading data from the table

Instructor Notes:

DML statements – Manipulates data in the database

Data Manipulation Language

- INSERT - Adds a new row to a table
- UPDATE - Changes existing data in a table
- DELETE - Removes rows from a table
- MERGE- Merges the data

Instructor Notes:

Reinforce the concept:
If we have to insert into column_1 value explicitly into this table then we need to SET IDENTITY_INSERT ON

Using INSERT

- INSERT statement adds one or more new rows to a table

  INSERT [INTO] table or view [(column_list)] data_values

```
INSERT INTO MyTable (PriKey, Description)
VALUES (123, 'A description of part 123.')
GO
```

- INSERT using SELECT statement

```
INSERT INTO MyTable (PriKey, Description)
SELECT ForeignKey, Description
FROM SomeTable
GO
```

The INSERT statement adds one or more new rows to a table. In a simplified treatment, INSERT has the following form:
INSERT [INTO] table_or_view [(column_list)] data_values
The INSERT statement inserts data_values as one or more rows into the specified table or view. column_list is a list of column names, separated by commas, that can be used to specify the columns for which data is supplied. If column_list is not specified, all the columns in the table or view receive data.
When column_list does not specify all the columns in a table or view, either the default value, if a default is defined for the column, or NULL is inserted into any column that is not specified in the list. All columns that are not specified in the column list must either allow for null values or have a default value assigned.
INSERT statements do not specify values for the following types of columns because the SQL Server Database Engine generates the values for these columns: 1.Columns with an IDENTITY property that generates the values for the column.2.Columns that have a default that uses the NEWID function to generate a unique GUID value.

```
CREATE TABLE dbo.T1
(                   column_1 int IDENTITY, column_2 varchar(30)
                    CONSTRAINT default_name DEFAULT ('my column default'))
INSERT INTO dbo.T1 (column_2) VALUES ('Explicit value');
INSERT INTO T1 DEFAULT VALUES;
```

If we have to insert into column_1 value explicitly into this table then we need to SET IDENTITY_INSERT ON

```
SET IDENTITY_INSERT dbo.T1 ON
INSERT INTO dbo.t1 (column_1,column_2) values(3,'some data');
```

Instructor Notes:

UPDATE statement

- UPDATE statement can change data values in single rows, groups of rows, or all the rows in a table or view

```
UPDATE products
SET unitprice=unitprice*1.1
WHERE categoryid=2
GO
```

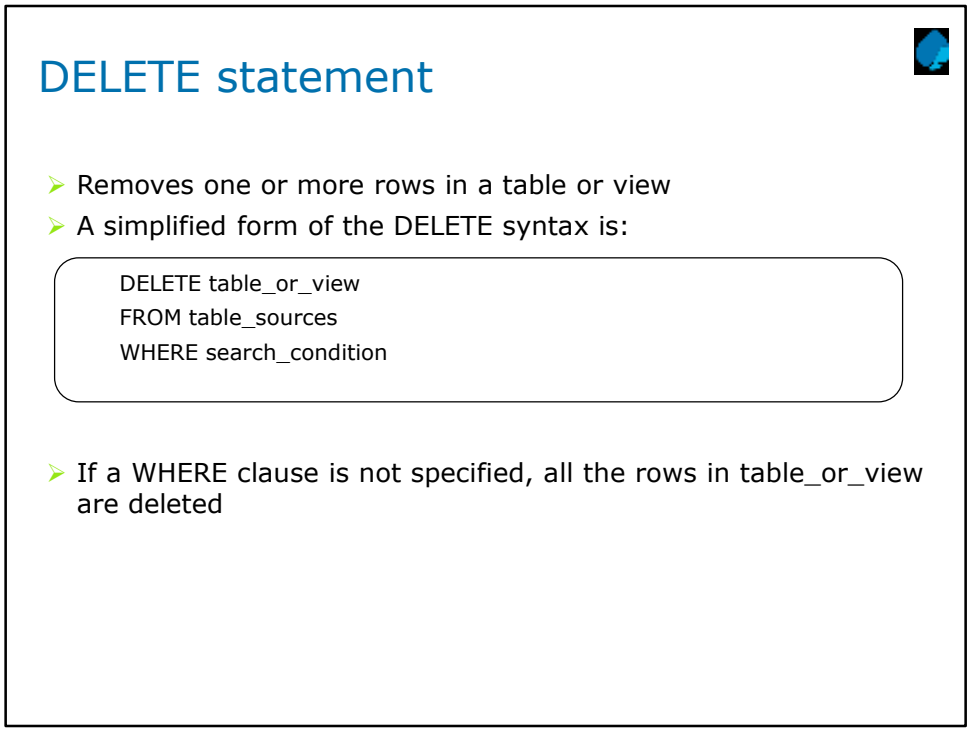
The UPDATE statement can change data values in single rows, groups of rows, or all the rows in a table or view. It can also be used to update rows in a remote server. An UPDATE statement referencing a table or view can change the data in only one base table at a time.
The UPDATE statement has the following major clauses:

SET
Contains a comma-separated list of the columns to be updated and the new value for each column, in the form *column_name* = *expression*. The value supplied by the expressions includes items such as constants, values selected from a column in another table or view, or values calculated by a complex expression.

FROM
Identifies the tables or views that supply the values for the expressions in the SET clause, and optional join conditions between the source tables or views.

WHERE
Specifies the search condition that defines the rows from the source tables and views that qualify to provide values to the expressions in the SET clause.

Instructor Notes:



The DELETE statement removes one or more rows in a table or view.
A simplified form of the DELETE syntax is:

```
DELETE table_or_view
 FROM table_sources
 WHERE search_condition
```

The parameter table_or_view names a table or view from which the rows are to be deleted. All rows in table_or_view that meet the qualifications of the WHERE search condition are deleted. If a WHERE clause is not specified, all the rows in table_or_view are deleted. The FROM clause specifies additional tables or views and join conditions that can be used by the predicates in the WHERE clause search condition to qualify the rows to be deleted from table_or_view. Rows are not deleted from the tables named in the FROM clause, only from the table named in table_or_view.
Any table that has all rows removed remains in the database. The DELETE statement deletes only rows from the table; the table must be removed from the database by using the DROP TABLE statement.

Instructor Notes:

MERGE statement (SQL Server 2008 onwards)

- In a typical database application, quite often you need to perform INSERT, UPDATE and DELETE operations on a TARGET table by matching the records from the SOURCE table
- To accomplish this, In previous versions of SQL Server, we had to write separate statements to INSERT, UPDATE, or DELETE data based on certain conditions
- Though it seems to be straight forward at first glance, but it becomes cumbersome when you have do it very often or on multiple tables
- Even the performance degrades significantly with this approach
- Now you can use MERGE SQL command to perform these operations in a single statement
- Using MERGE statement we can include the logic of such data modifications in one statement that even checks when the data is matched then just update it and when unmatched then insert it

MERGE Statement

In a typical database application, quite often you need to perform INSERT, UPDATE and DELETE operations on a TARGET table by matching the records from the SOURCE table. For example, a products dimension table has information about the products; you need to sync-up this table with the latest information about the products from the source table. You would need to write separate INSERT, UPDATE and DELETE statements to refresh the target table with an updated product list or do lookups. Though it seems to be straight forward at first glance, but it becomes cumbersome when you have do it very often or on multiple tables, even the performance degrades significantly with this approach.

With SQL Server 2008, now you can use MERGE SQL command to perform these operations in a single statement. The MERGE statement basically merges data from a source result set to a target table based on a condition that you specify and if the data from the source already exists in the target or not. The new SQL command combines the sequence of conditional INSERT, UPDATE and DELETE commands in a single atomic statement, depending on the existence of a record.

Instructor Notes:

MERGE statement

- The Merge statement comprises Five clauses
  - MERGE - Used to specify the Target tables to be inserted/updated/deleted
  - USING - Used to specify the Source table
  - ON - Used to specify the join condition on source and target tables
  - WHEN - Used to specify the action to be taken place based on the result of the ON clause
  - OUTPUT - Used to return the value of Insert/Update/Delete operations using the INSERTED / DELETED magic tables
- MERGE Statement - Syntax

```
MERGE <target_table> [AS TARGET]
USING <table_source> [AS SOURCE]
ON <search_condition>
[WHEN MATCHED THEN <merge_matched> ]
[WHEN NOT MATCHED [BY TARGET] THEN
<merge_not_matched> ]
[WHEN NOT MATCHED BY SOURCE THEN <merge_ matched> ];
```

The Clauses used in MERGE Statement

The Merge statement uses two tables (Source and Target tables). The Source table is specified with USING clause. The Target table is specified with MERGE INTO clause.
Merge statement is similar to OUTER joins. We can use condition with Merge statement like WHEN MATCHED THEN, WHEN NOT MATCHED [BY TARGET] THEN , WHEN NOT MATCHED BY SOURCE THEN.
If the record is already there in target table then we can update the data, If its not there in target table then we an insert.
The OUTPUT clause can be used with Merge statement to return an output as INSERTED / DELETED like magic tables. We can use $action function to identity that what kind of action taken place like INSERT / UPDATE / DELETE.

Instructor Notes:

MERGE statement

- MERGE Statement - Example

```
MERGE INTO dbo.Customers AS TGT
USING dbo.CustomersStage AS SRC
ON TGT.custid = SRC.custid
WHEN MATCHED THEN
UPDATE SET
TGT.companyname = SRC.companyname,
TGT.phone = SRC.phone,
TGT.address = SRC.address
WHEN NOT MATCHED THEN
INSERT (custid, companyname, phone, address)
VALUES (SRC.custid, SRC.companyname, SRC.phone,
SRC.address)
WHEN NOT MATCHED BY SOURCE THEN
DELETE
OUTPUT
$action, deleted.custid AS del_custid, inserted.custid AS
ins_custid;
```

Instructor Notes:

MERGE statement

- The given example does the following
  - MERGE statement defines the Customers table as the target for the modification
  - CustomersState table as the source
  - The MERGE condition matches the custid attribute in the source with the custid attribute in the target
  - When a match is found in the target, the target customer's attributes are overwritten with the source customer attributes
  - When a match is not found in the target, a new row is inserted into the target, using the source customer attributes
  - When a match is not found in the source, the target customer row is deleted

Instructor Notes:

Demo

- Using MERGE Statement

Instructor Notes:

DCL Statements

- DCL (Data Control Language)
  - DCL Statement is used for securing the database
  - DCL Statement control access to database
  - DCL statements supported by T-SQL are GRANT , REVOKE and DENY

Instructor Notes:

DCL Statements

- GRANT authorizes one or more users to perform an operation or a set of operations on an object.
  - GRANT PRIVILEGES ON object-name TO endusers;
  - GRANT SELECT, UPDATE ON My_table TO some_user, another_user;
  - GRANT INSERT (empno, ename, job) ON emp TO endusers;
- REVOKE removes or restricts the capability of a user to perform an operation or a set of operations.
  - REVOKE PRIVILEGES ON object_name FROM endusers
  - REVOKE INSERT, DELETE ON emp FROM enduser
- DENY denies permission to a user/group , even though he may belong to a group/role having the permission
  - DENY SELECT ON Employees TO Ruser1

Instructor Notes:

None

Summary

- In this lesson, you have learnt:
  - Different datatypes to store numeric, character, monetary and date time data
  - New data types in SQL Server 2008 - Hierarchyid & Spatial
  - Creating UDTS
  - Creating Tables ,Manipulating Data and Assigning and revoking privileges.
  - MERGE provides an efficient way to perform multiple DML operations
  - Implement constraints like – Primary key, Foreign key, check, default and unique
  - Alter table to change structure, enable/disable constraints

Instructor Notes:

Review Question

- Question 1: To create a user defined data type we use _____.
- Question 2: If we want the column data to have unique values with null then we should use _____ constraint.
- Question 3: _____ and _____ are objects like constraints that are bound to the table.
- Question 4: _____ option is used to create a constraint that should not be temporarily checked for.
- Question 5: Use _____ as a data type to create tables with a hierarchical structure.
- Question 6: _____ Used to return the value of Insert/Update/Delete operations using the INSERTED / DELETED magic tables