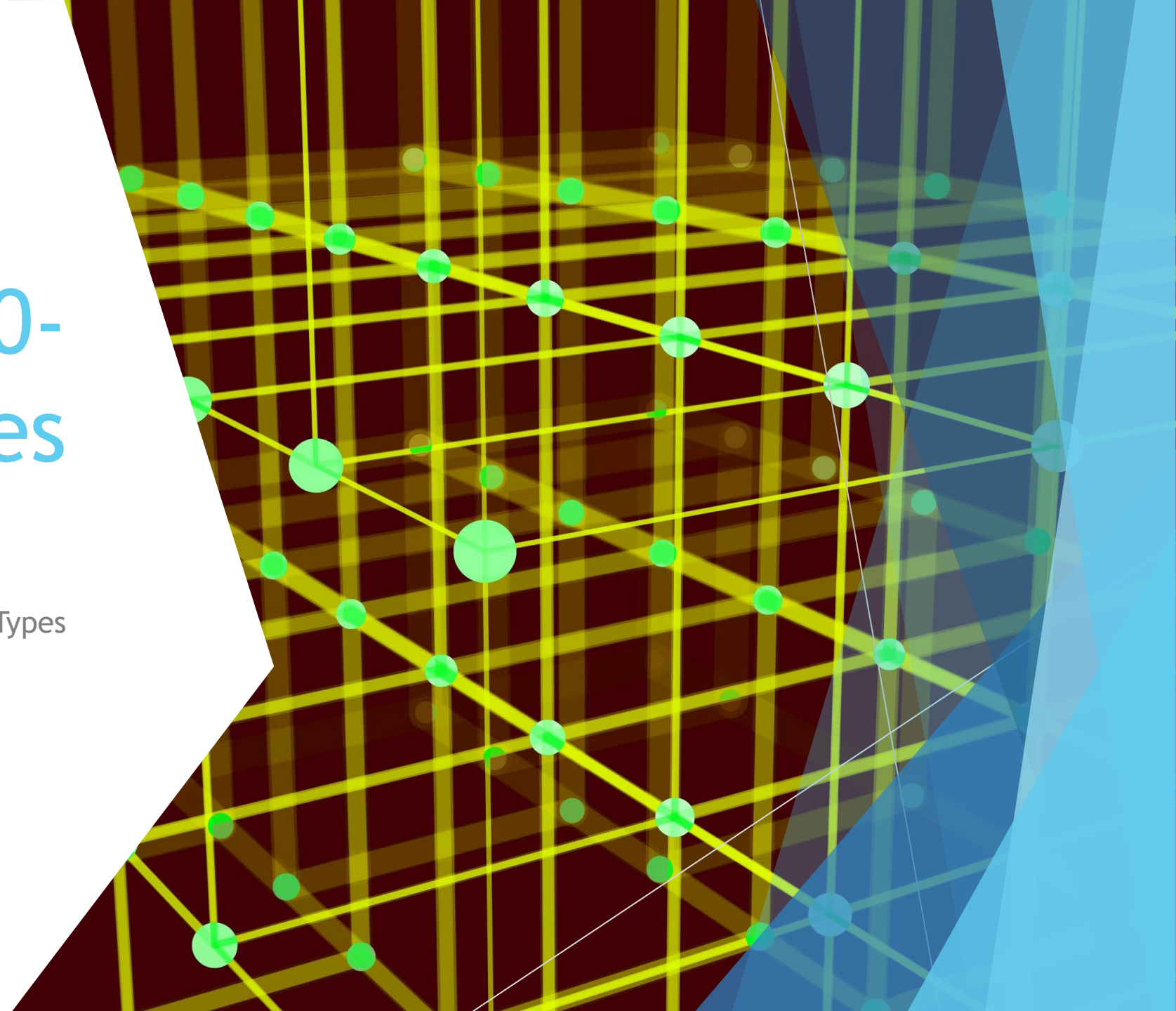
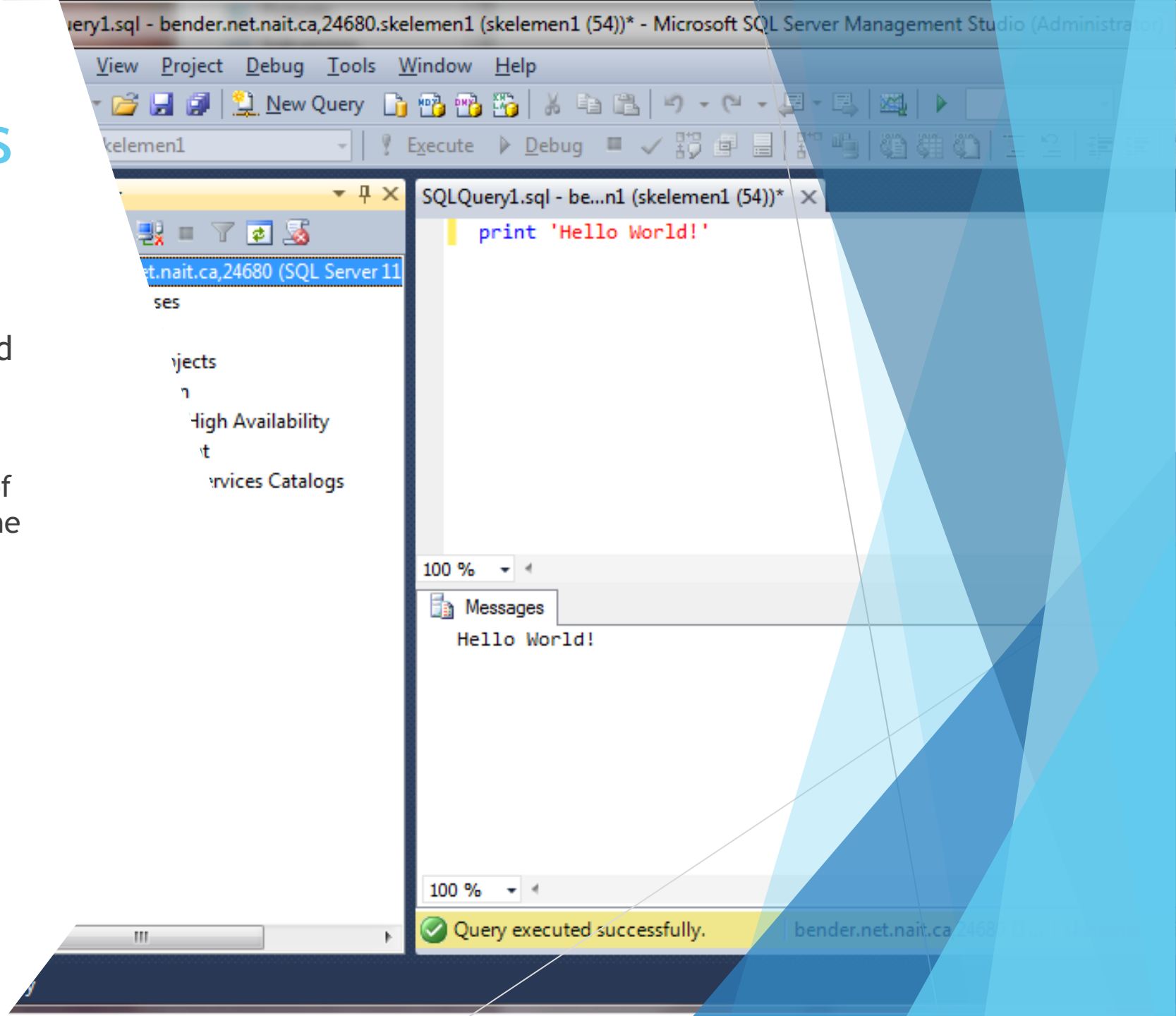
CMPE 2400- Databases

SQL Data Types



Printing Messages to the User

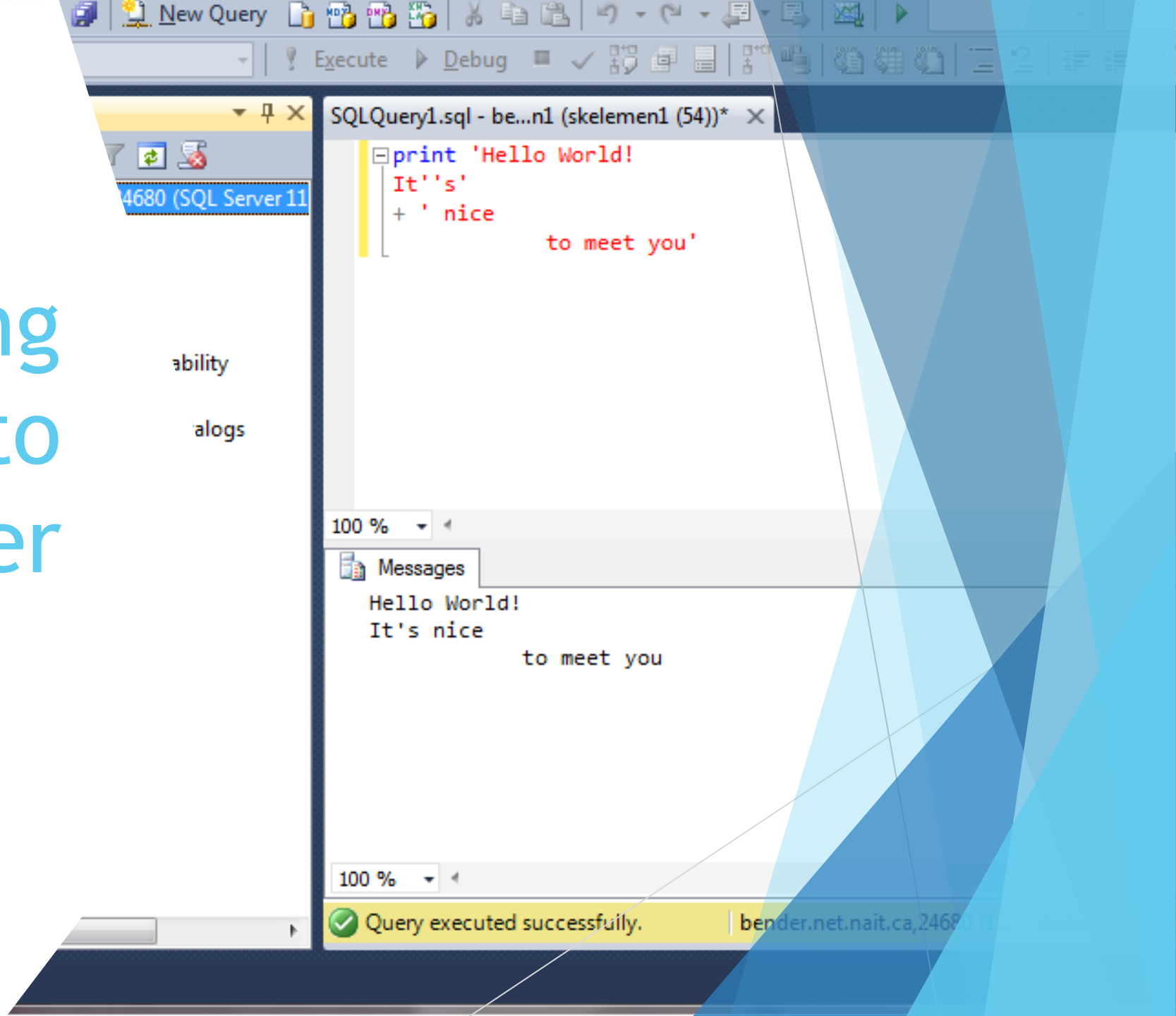
- The print keyword may be used to send a message to the messages results pane.
  - You may toggle the visibility of the messages pane by using the hotkey combination Ctrl+R



Printing Messages to the User

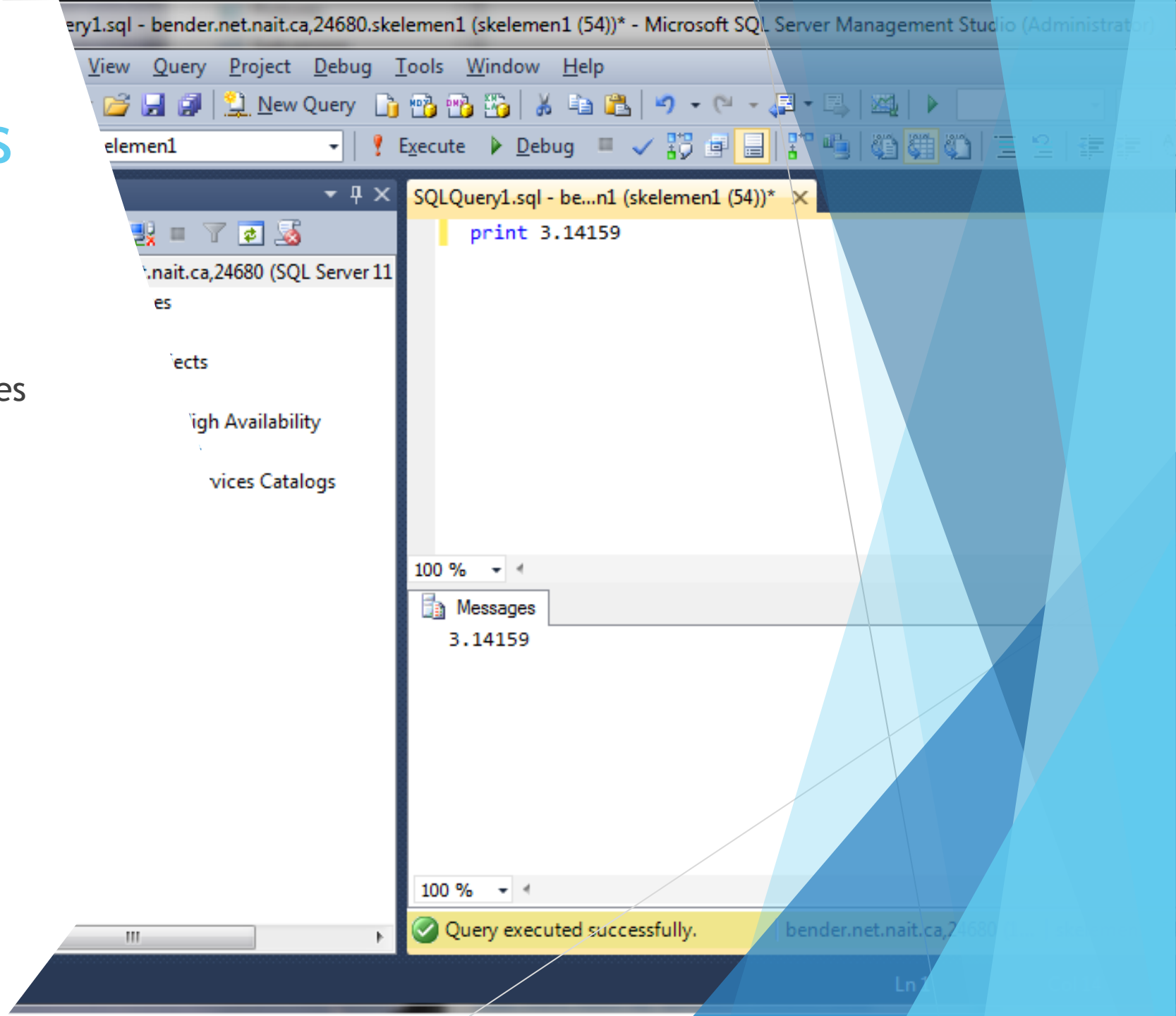
- String literals in SQL are evaluated similarly to verbatim strings in C#, thus:
  - All whitespace will be preserved, including tabs and newlines.
  - Two or more string literals may be concatenated using the + operator.
  - A single quote is displayed using two single quotes side-by-side

Printing Messages to the User

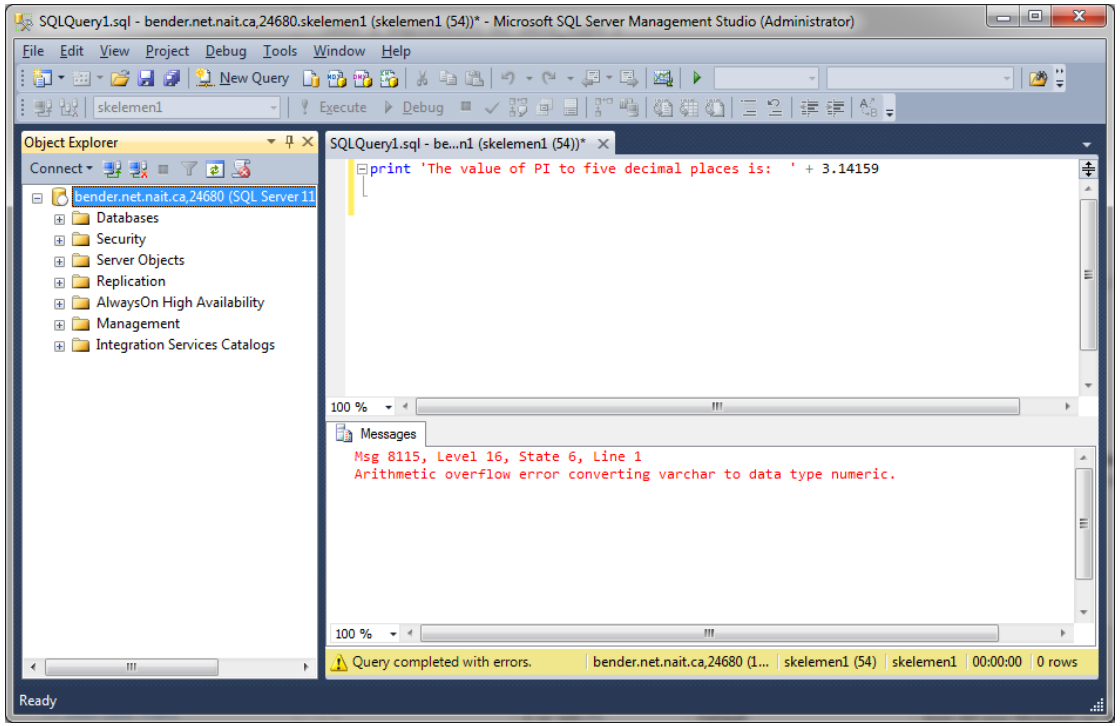


Printing Messages to the User

- Numerical literals may also be printed directly to the messages pane.



Printing Messages to the User



- Unfortunately, attempting to combine the two will lead to disaster.

Printing Messages to the User

- To avoid the message in the previous operation you may of course choose to use two separate print statements, but this does not really solve the problem.
  - It is better to convert the number into a compatible type, just as in C#, but we must do so manually in SQL. Two built in functions may be used:
    - CAST – Preferred as it is part of the ANSI specification
    - CONVERT – Microsoft SQL Server specific but still useful when you wish to convert to a specific string format from a date or floating point type. See the link below to explore.

      http://msdn.microsoft.com/en-us/library/aa226054%28v=sql.80%29.aspx

CAST

- CAST Syntax

  cast (*expression* as *dataType* [*(length)*])

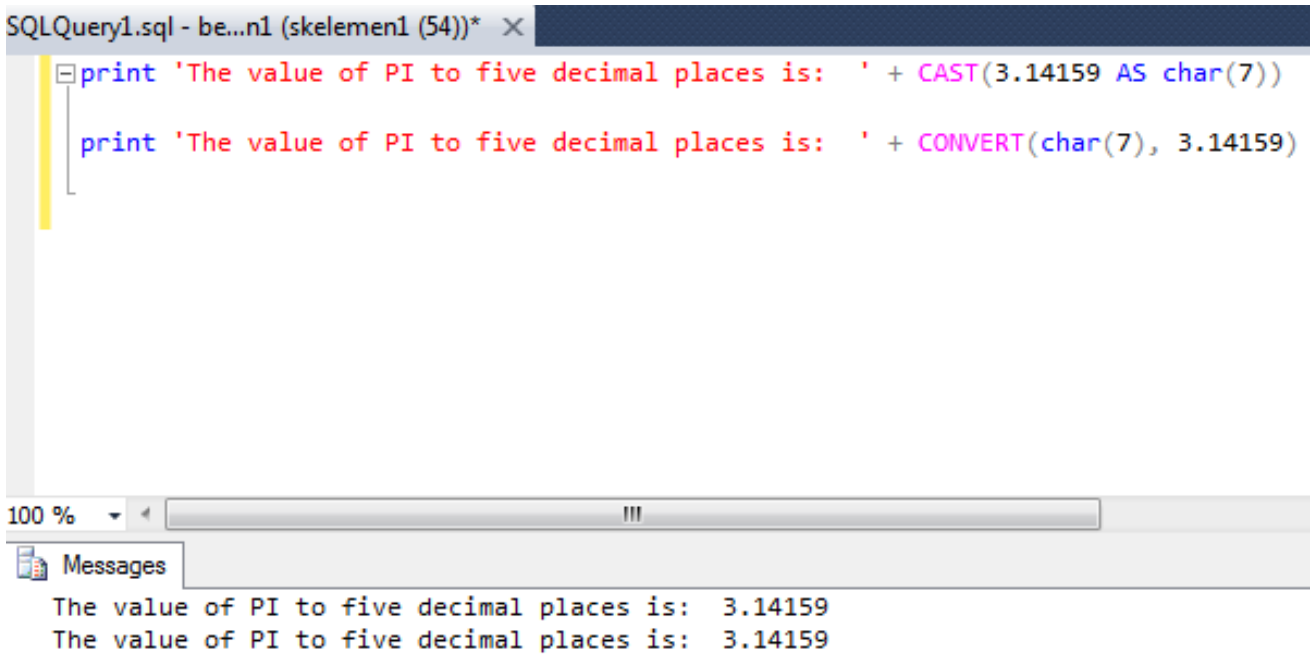
  - where:
    - *expression* is a string or numeric literal, variable, or the result of a calculation or function.
    - *dataType* is the data type into which you want to convert the expression (make sure it is a compatible type)
    - *length* is an optional parameter used when you are trying to convert into a string type.

CONVERT

- CONVERT Syntax

  convert (*dataType* [*(length)*], *expression*[, *style*])

  - where:
    - *expression* is a string or numeric literal, variable, or the result of a calculation or function.
    - *dataType* is the data type into which you want to convert the expression (make sure it is a compatible type)
    - *length* is an optional parameter used when you are trying to convert into a string type.
    - *style* is an integer value indicating the format to use when converting a date or floating point type into a string type.



Printing Messages to the User

The above shows both cast and convert being used to fix our previous error.

Data Types

- Exact Numerics
- Approximate Numerics
- Date and Time
- Character Strings
- Unicode Character Strings
- Binary Strings
- Other (Only the *table* type will be covered later in the course)

Exact Numerics

- This category includes types where every number in the range of the type may be written down in perfect detail.
  - The integer types.
    - Converting any *non-zero* number into a bit will be evaluated as a 1, or true.
    - The string *true* converts to a 1 and *false* to a 0.

Data Type	Range	Storage Size	C# Equivalency
bigint	-2^{63} -> $2^{63}-1$	8 bytes	long
int	-2^{31} -> $2^{31}-1$	4 bytes	int
smallint	-2^{15} -> $2^{15}-1$	2 bytes	short
tinyint	0 -> $2^{8}-1$	1 byte	byte
bit	0 -> 1	1-8 bits = 1 byte, etc.	bool

Exact Numerics

- This category includes types where every number in the range of the type may be written down in perfect detail.
  - The decimal types.
    - The *decimal* and *numeric* types are interchangeable and explained further on the next slide.

Data Type	Range	Storage Size	C# Equivalency
decimal	$-10^{38}+1$ -> $10^{38}-1$	See Next Slide	decimal
numeric	$-10^{38}+1$ -> $10^{38}-1$	See Next Slide	decimal
money	-922,337,203,685,477.5808 -> 922,337,203,685,477.5807	8 bytes	decimal
smallmoney	- 214,748.3648 -> 214,748.3647	4 bytes	decimal

Exact Numerics

- The *numeric* and *decimal* data types have the following syntax.
  - Remember that they are interchangeable…

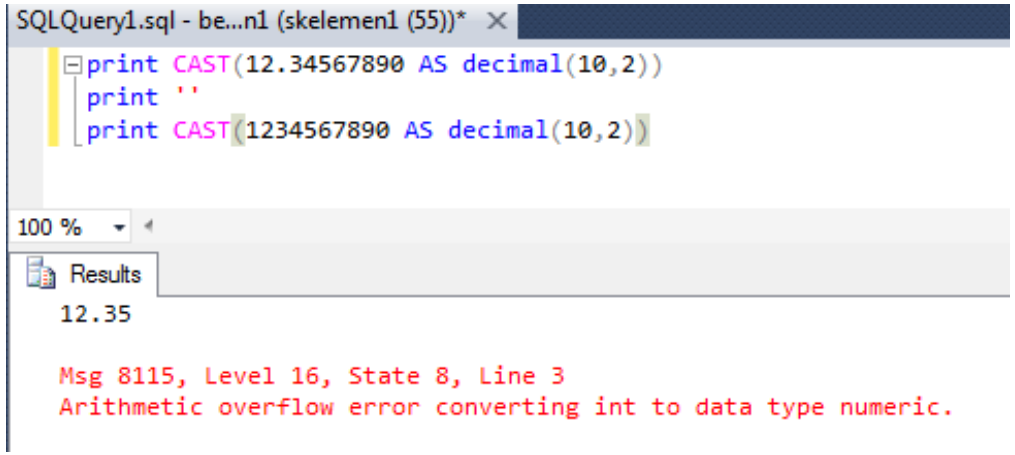
    numeric [*(precision* [, *scale*] *)*]

    where:
    - *precision* is the total number of digits used to represent the number. Values are 1 to 38. Default is 18.
    - *scale* is the number of digits of precision to the right of the decimal place. Values are 0 to *precision*. Default is 0.

  - Storage depends on *precision*:

Precision	Storage Size
1 -> 9	5
10 -> 19	9
20 -> 28	13
29 -> 38	17

Exact Numerics



Great care must be taken when converting values as not all incorrect conversions will cause an error.

- Note that the first value was stored after truncation which equates to a loss of information.

Approximate Numerics

- This category contains two types used to store much smaller and much larger numbers, but a loss of accuracy is experienced.
  - The number is stored as the combination of the *mantissa* and *exponent* of a scientific notation value.
    - For example, in the number 1.234567 E-15, the *mantissa* is 1.234567, and the *exponent* is -15.

Data Type	Range	Storage Size	C# Equivalency
float	- 1.79 E+308 -> -2.23 E-308, 0 and 2.23 E-308 -> 1.79 E+308	See Next Slide	double
real	- 3.40 E+38 -> -1.18 E-38, 0 and 1.18 E-38 -> 3.40 E+38	4 bytes	float

Approximate Numerics

- The *float* data type has the following syntax.

  float [*(n)*]

  where:

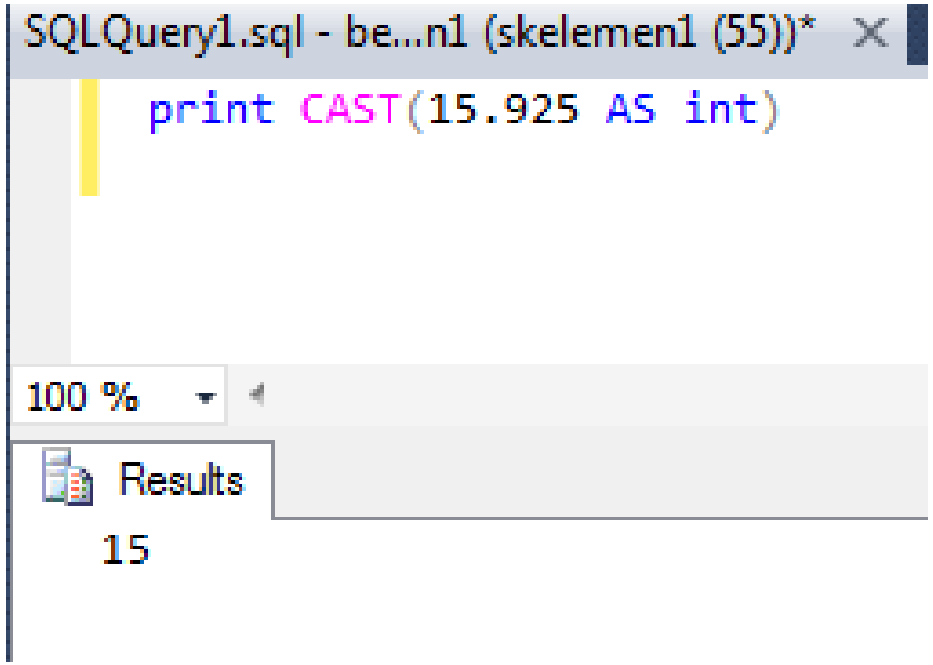
  - *n* is the total number of bits used to store the *mantissa* of the number. Values are 1 to 53. Default is 53.

- Storage depends on *n*:

n	Precision	Storage Size
1 -> 24	7 digits	4 bytes
25 -> 53	15 digits	8 bytes

Approximate Numerics

When converting from approximate types into integral types, excess information is truncated.



Date and Time

- Many different date and time types exist to offer the database designer the ability to choose an appropriate data type for their specific needs.
  - For all types, the date component always has a default value of January 1, 1900. Time components are always set to exactly midnight.
  - *datetimeoffset* stores a value from -14:00 to 14:00 that is the time shift from UTC. Default is 0.

Data Type	Date Range	Accuracy (max.)	Storage Size	C# Equivalency
date	January 1, 0001 -> December 31, 9999	1 day	3 bytes	DateTime
smalldatetime	January 1, 1900 -> June 6, 2079	1 minute	4 bytes	DateTime
datetime	January 1, 1753 -> December 31, 9999	3.33 milliseconds	8 bytes	DateTime
datetime2	January 1, 0001 -> December 31, 9999	100 nanoseconds	See Next Slide	DateTime
datetimeoffset	January 1, 0001 -> December 31, 9999	100 nanoseconds	See Next Slide	DateTimeOffset
time	Not Applicable	100 nanoseconds	See Next Slide	TimeSpan

Date and Time

- The *time, datetime2, and datetimeoffset* types allow you to choose the number of decimal places of fractional seconds. The syntax is:
  - Again the different types are interchangeable for this syntax…

    datetimeoffset [*(precision)*]

    where:
    - *precision* is the number of decimal places of fractional seconds. Values are 0 to 7. Default is 7.
  - Storage depends on *precision:*

Data Type	Precision	Storage Size
datetime2	0 -> 2	6 bytes
datetime2	3 -> 4	7 bytes
datetime2	5 -> 7	8 bytes
datetimeoffset	0 -> 2	8 bytes
datetimeoffset	3 -> 4	9 bytes
datetimeoffset	5 -> 7	10 bytes
time	0 -> 2	3 bytes
time	3 -> 4	4 bytes
time	5 -> 7	5 bytes

Exercise

- Write suitable statements, using the getdate() function (with appropriate cast), to display:
  1. The current time, giving fractional seconds to 4 decimal places
  2. The current date only (time not displayed)
  3. The current date and time, giving the fractional seconds to 5 decimal places

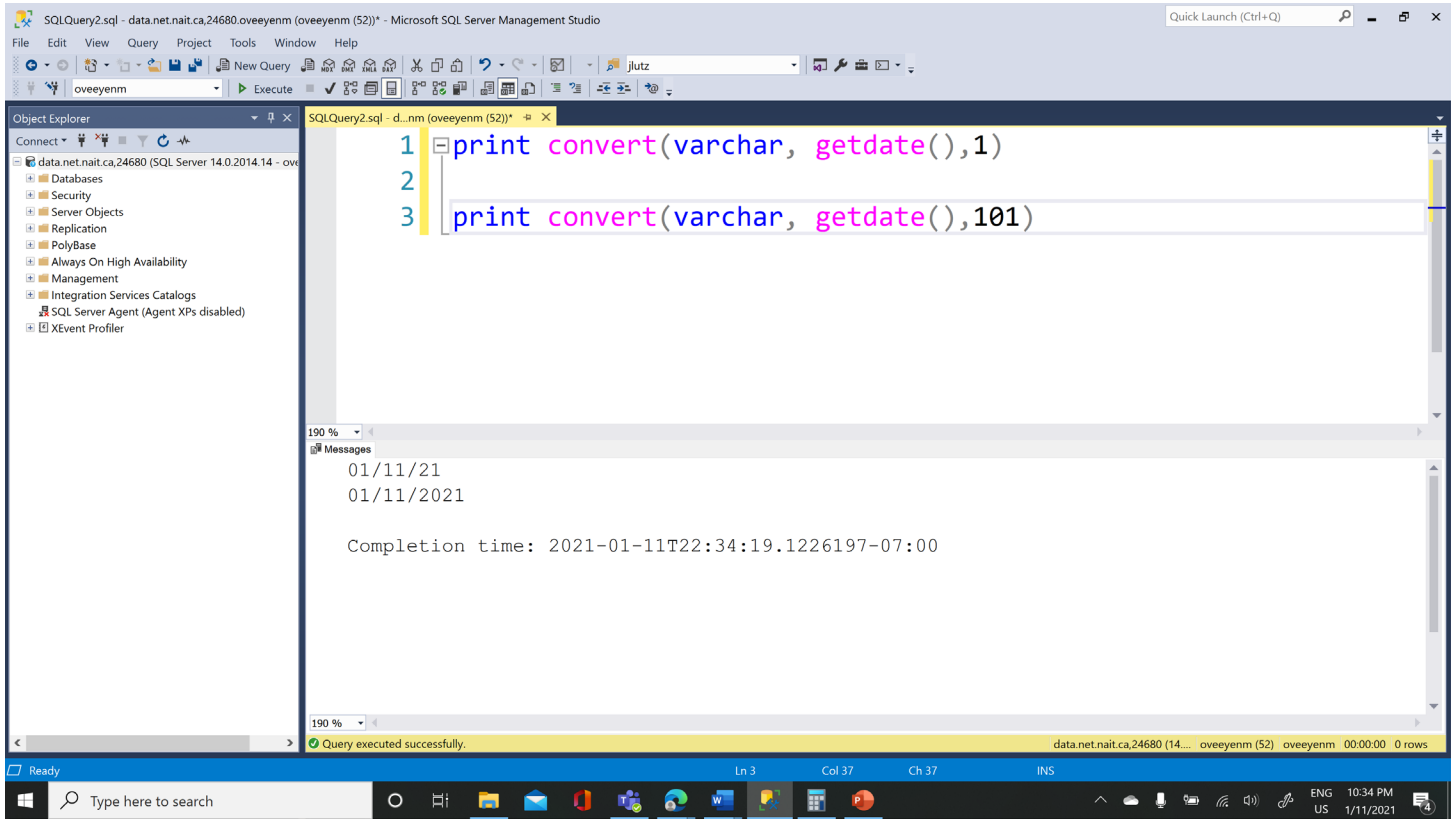
Displaying Date and time using Convert

When displaying a date as a string, we use convert with the style parameter to display the date and time in different ways.

Note that the style value is an integer. If the value is 100 or above, it uses 4 digits for year. Otherwise it uses 2 digits for year.

Some examples are given in the next slide

Displaying Date and time using Convert



Exercise

- Using convert and getdate(), display the current date (and time where applicable) using each of the following styles:
  - 1
  - 101
  - 3
  - 103
  - 13
  - 113
  - 20
  - 120

Date and Time

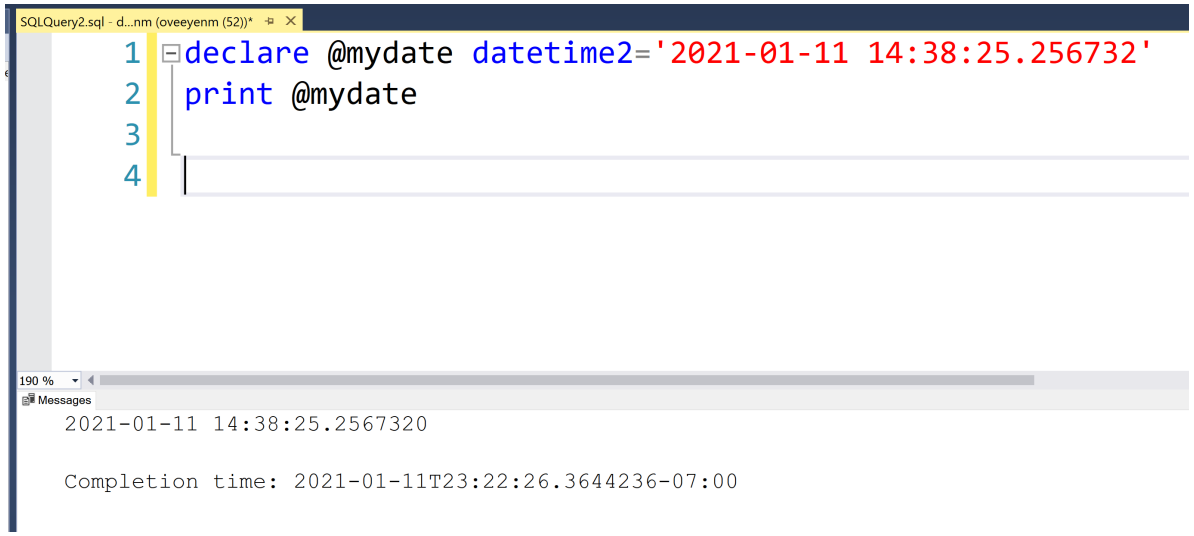
- In SQL, it is possible to construct a time value using a string literal. There are often several formats that will be recognized by a database engine, but the following conform to ANSI, including the displayed spaces.
  - *smalldatetime* and *datetime*

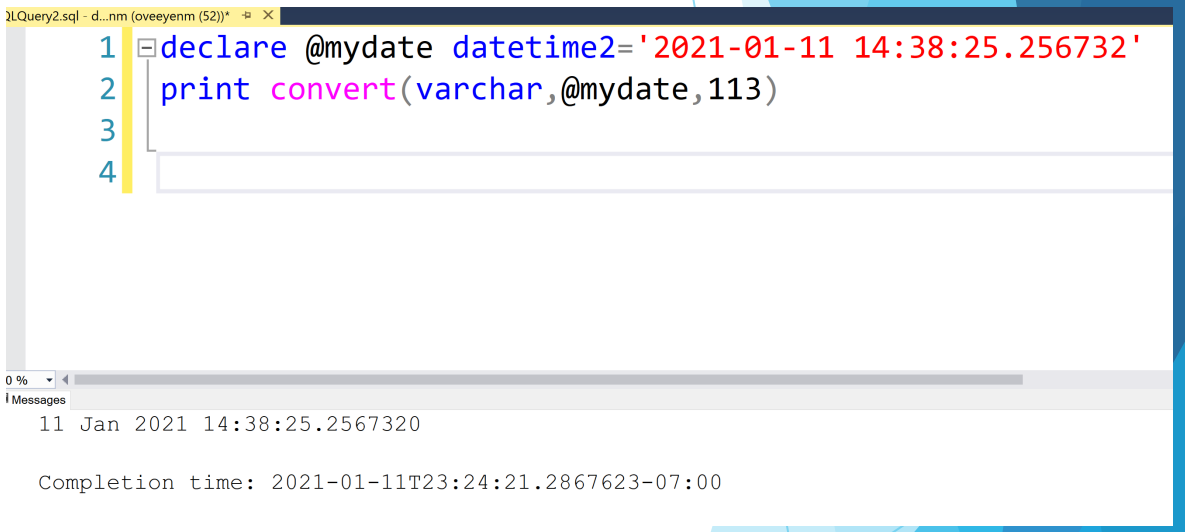
    yyyy-mm-dd hh:mm[:ss[.fff]]

  - date, time, *datetime2*, and *datetimeoffset*

    yyyy-mm-dd hh:mm[:ss[.fffffff]] [+/-hh:mm]

Date and Time

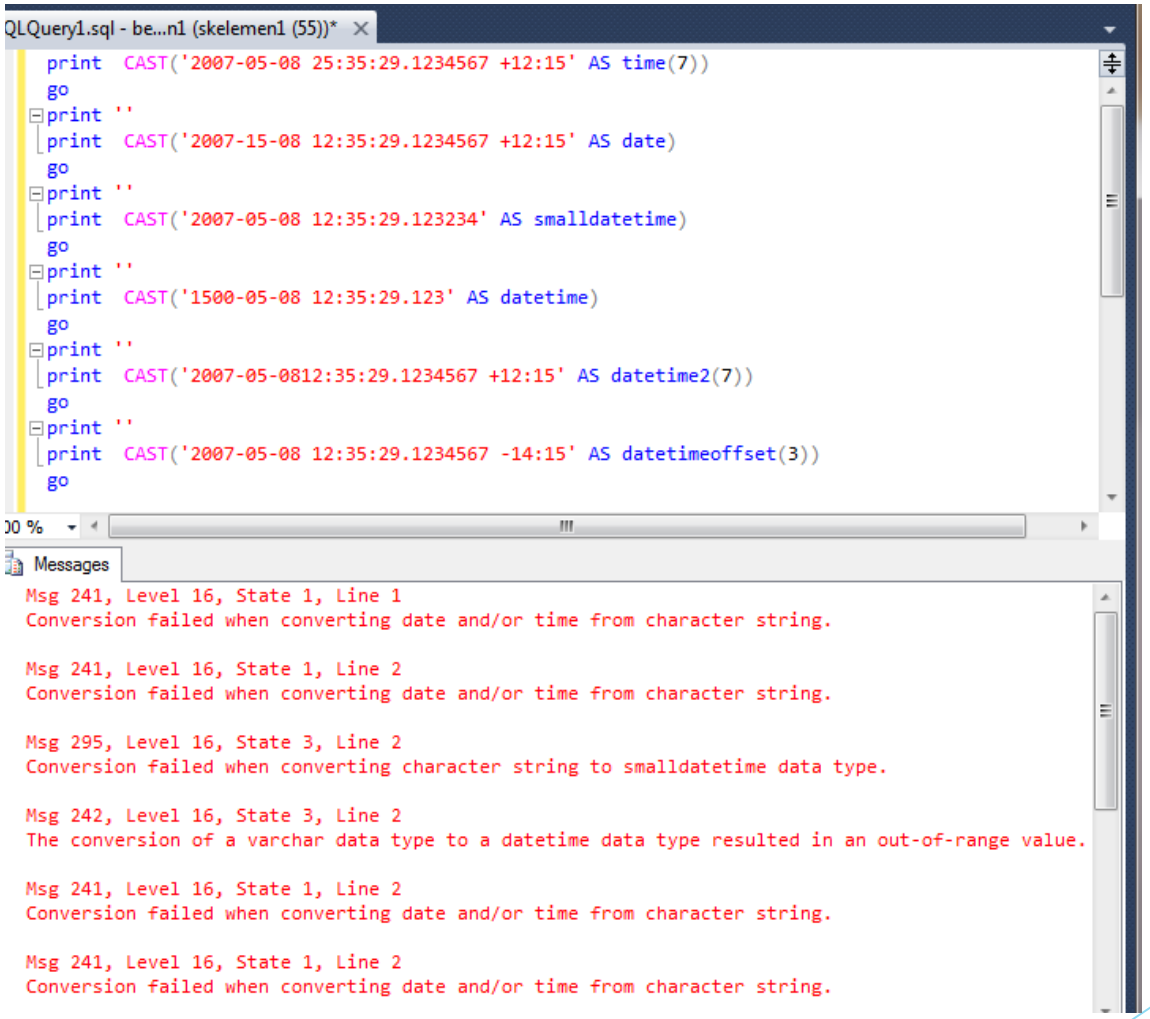




Date and Time

- Converting date and time values can be tricky, so be very careful to use correct values in your literals.
  - Examine the string literals in the examples on the following slide and see if you can spot the errors.

Date and Time



Character Strings

- These data types represent strings constructed using the ASCII character set.
  - You should only use these strings when you are 100% positive that your data requirements will be forever restricted to the English language.
    - The *char* type is used when you know the exact size of the string to be stored, otherwise use *varchar*.
    - Though the *text* type has been defined below, it has been deprecated and is no longer supported. Use *varchar(max)* instead.

Data Type	Length Range	Storage Size	C# Equivalency
char	1 -> 8000 characters	*length* bytes	None
varchar	1 -> 8000 characters or 2^{31}-1 characters	*length* + 2 bytes	None
text (do not use)	2^{31}-1 characters	*length* + 2 bytes	None

Character Strings

- The syntax for the character types is as follows:

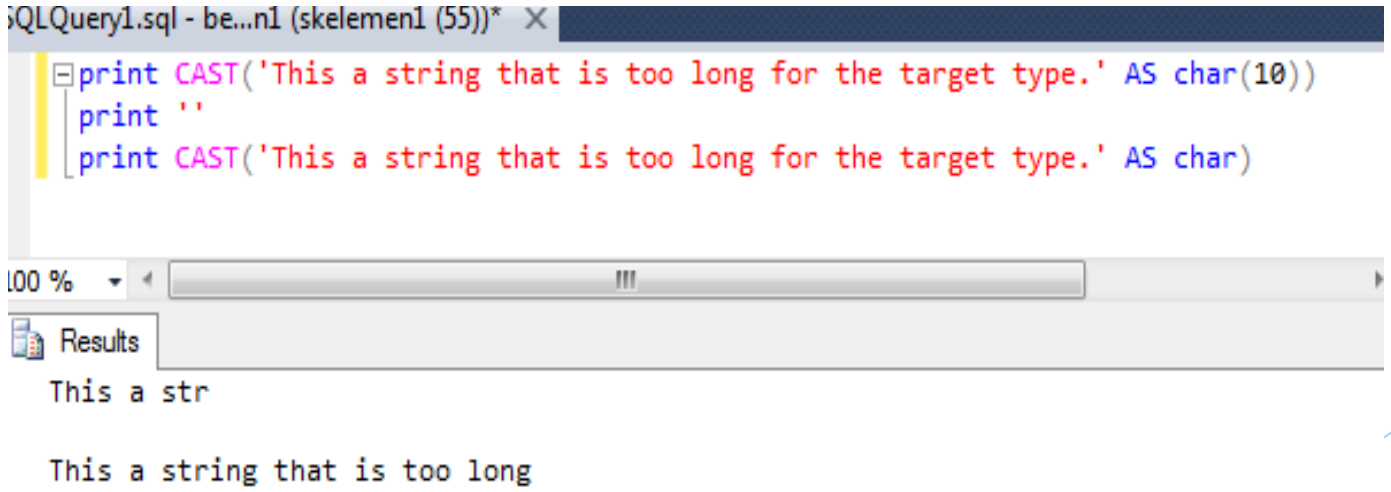
char [*(length)*] *varchar* [*(length* | *max)*] *text*

where:

- *length* is the number of characters in the string. Values are 1 to 8000. Default value is 1 when specifying variable sizes and 30 when using CAST or CONVERT. When the *max* modifier is used, 2^{31}-1 characters are permitted.

Character Strings

- Be careful when converting into string format. If you are lucky you will receive an error, but usually truncation will occur which can lead to very difficult debugging.



Unicode Character Strings

- These data types represent strings constructed using the Unicode character set.
  - These data types should be used when you are uncertain whether you will need to store values from languages other than English.
    - The *nchar* type is used when you know the exact size of the string to be stored, otherwise use *nvarchar*.
    - Though the *ntext* type has been defined below, it has been deprecated and is no longer supported. Use *nvarchar(max)* instead.

Data Type	Length Range	Storage Size	C# Equivalency
nchar	1 -> 4000 characters	(*length* * 2) bytes	string or char[ ]
nvarchar	1 -> 4000 characters or 2^{30}-1 characters	(*length* * 2) + 2 bytes	string or char[ ]
ntext (do not use)	2^{30}-1 characters	(*length* * 2) + 2 bytes	None

Unicode Character Strings

- The syntax for the Unicode character types is as follows:

nchar [*(length)*] *nvarchar* [*(length* | *max)*] *ntext*

where:

- *length* is the number of characters in the string. Values are 1 to 4000. Default value is 1 when specifying variable sizes and 30 when using CAST or CONVERT. When the *max* modifier is used, 2^{30}-1 characters are permitted.

Binary Strings

- As you have probably guessed, these types are for storing binary data.
  - Examples here would include documents, photos, video, and sound files.
    - The *binary* type is used when you know the exact size of the string to be stored, otherwise use *varbinary*.
    - Though the *image* type has been defined below, it has been deprecated and is no longer supported. Use *varbinary(max)* instead.

Data Type	Length Range	Storage Size	C# Equivalency
binary	1 -> 8000 bytes	*length* bytes	byte[ ]
varbinary	1 -> 8000 bytes or 2^{31}-1 bytes	*length* + 2 bytes	byte[ ]
image (do not use)	2^{31}-1 bytes	*length* + 2 bytes	None

Binary Strings

- The syntax for the characters types is as follows:

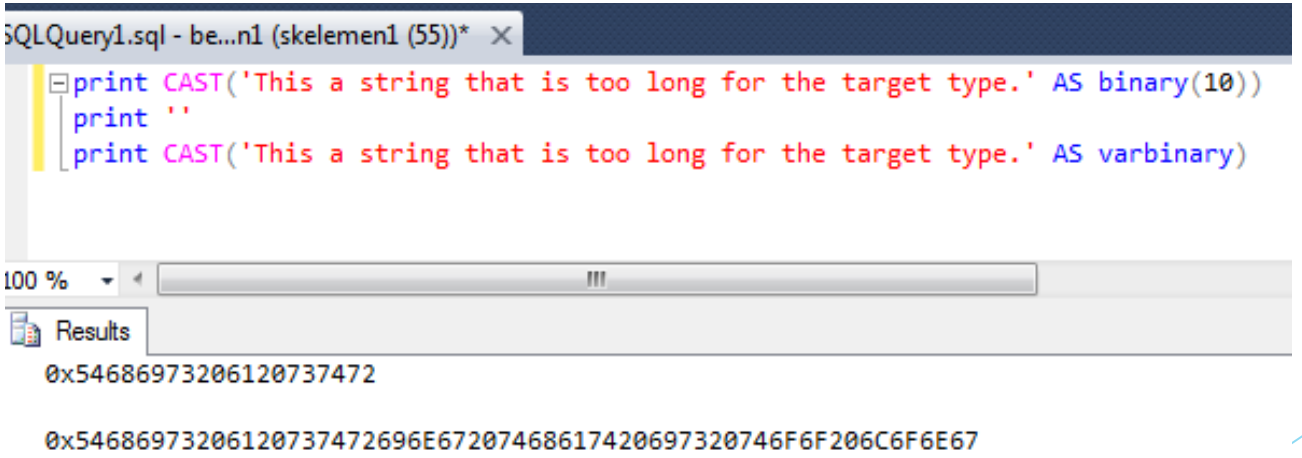
binary[*(length)*] *varbinary*[*(length* | *max)*] *image*

where:

- *length* is the number of characters in the string. Values are 1 to 8000. Default value is 1 when specifying variable sizes and 30 when using CAST or CONVERT. When the *max* modifier is used, 2^{31}-1 characters are permitted.

Binary Strings

- Be careful when converting into binary format. Truncation may occur which can lead to very difficult debugging.
  - Remember that there are 2 hexadecimal digits in 1 byte.



Defining Variables

declare *@name* *dataType* [= *value*]

where:

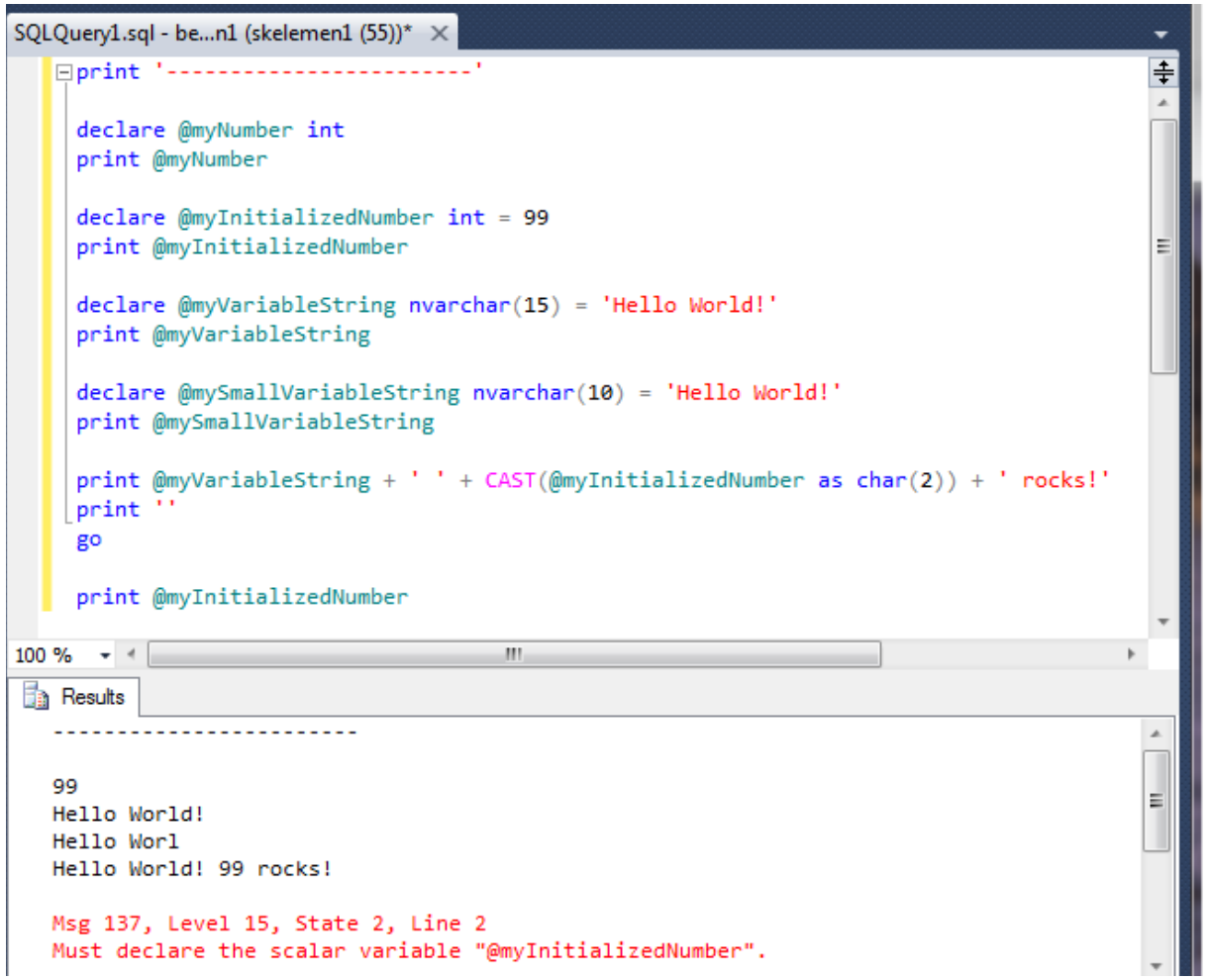
- *declare* is the keyword which begins the variable definition
- *@name* is the variable's name
- *dataType* is any dataType defined in the previous section.
- = *value* is the optional initialization part of the syntax. Default value is null.

Defining Variables

- You may define multiple variables using a single declare statement, even if they are not the same type, but all variable names must adhere to the following rules:
  - The @ must be the first character (only for SQL Server), but do not use @ as the second character.
  - Length must be between 1 and 128 characters.
    - Improper variable names will be frowned upon… greatly.
  - After the first @, the rest of the variable name may be composed of any Unicode 2.0 compliant character, or the symbols @, $, #, _.
  - The camelCase convention will be used after the @.

Defining Variables

- Examine the variable declarations below.
  - Note the error received from the print statement following the go keyword. This defines the end of the *batch or scope*.



Assigning to Variables

- Until recently, initializing a variable was always completed as an assignment after its definition. The syntax is as follows:

```
set @name = value
```

where:

- *@name* is the variable receiving the new value.
- *value* is the value being assigned to the variable.

Assigning to Variables

- A simple reassignment example:

```
declare @variableName int = 99
print @variableName

set @variableName = 1000
print @variableName
```

100 %

Results

```
99
1000
```

Basic Math Operators

Precedence Group	Operator List	Associativity
Override	**( )** (Parenthesis)	Left -> Right
Bitwise Inversion	**~** (Bitwise NOT)	Left -> Right
Multiplicative	***** (Multiply), **/** (Divide), **%** (Modulus)	Left -> Right
Additive	**+** (Positive), **-** (Negative), **+** (Add), **+** (Concatenate), **-** (Subtract), **&** (Bitwise AND), **|** (Bitwise OR), **^** (Bitwise XOR)	Left -> Right
Assignment	**=** (Assignment), **+=** (Add Equals), **-=** (Subtract Equals), ***=** (Multiply Equals), **/=** (Divide Equals), **%=** (Modulus Equals), **&=** (Bitwise AND Equals), **|** (Bitwise OR Equals), **^=** (Bitwise XOR Equals)	Right -> Left

Basic Math Operators

- Note: compound operators may not be chained in SQL!

```
declare @variableName int = 99

set @variableName += 1000 %= 4
print @variableName
```

0 %

Results

```
Msg 102, Level 15, State 1, Line 3
Incorrect syntax near '%='.
```