SQL Server 2017

Filter by title

Configuration
Conversion
CAST and CONVERT
PARSE
TRY_CAST
TRY_CONVERT
TRY_PARSE
Cryptographic
Cursor
Data type
Date and time

Download PDF

CAST and CONVERT (Transact-SQL)

04/13/2018 22 minutes to read Contributors all

APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

Please help improve SQL Server docs!

These functions convert an expression of one data type to another.

Example: Change the input datatype

Cast

```sql
SELECT 9.5 AS Original, CAST(9.5 AS int) AS int,
    CAST(9.5 AS decimal(6,4)) AS decimal;
```

Convert

```sql
SELECT 9.5 AS Original, CONVERT(int, 9.5) AS int,
    CONVERT(decimal(6,4), 9.5) AS decimal;
```

Here is the result set.

Original	int	decimal
9.5	9	9.5000

See the examples later in this topic.

Transact-SQL Syntax Conventions

Syntax

```sql
-- CAST Syntax:
CAST ( expression AS data_type [ ( length ) ] )
```

```
-- CONVERT Syntax:
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

Arguments

expression
Any valid expression.

data_type
The target data type. This includes **xml**, **bigint**, and **sql_variant**. Alias data types cannot be used.

length
An optional integer that specifies the length of the target data type. The default value is 30.

style
An integer expression that specifies how the CONVERT function will translate *expression*. For a style value of NULL, NULL is returned. *data_type* determines the range.

Return types

Returns *expression*, translated to *data_type*.

Date and Time Styles

For a date or time data type *expression*, *style* can have one of the values shown in the following table. Other values are processed as 0. Beginning with SQL Server 2012 (11.x), the only styles supported, when converting from date and time types to **datetimeoffset**, are 0 or 1. All other conversion styles return error 9809.

> ⓘ **Note**
>
> SQL Server supports the date format, in Arabic style, with the Kuwaiti algorithm.

Without century (yy) ([1])	With century (yyyy)	Standard	Input/Output ([3])
-	0 or 100 ([1,2])	Default for datetime and smalldatetime	mon dd yyyy hh:miAM (or PM)
1	101	U.S.	1 = mm/dd/yy 101 = mm/dd/yyyy

2	**102**	ANSI	2 = yy.mm.dd 102 = yyyy.mm.dd
3	**103**	British/French	3 = dd/mm/yy 103 = dd/mm/yyyy
4	**104**	German	4 = dd.mm.yy 104 = dd.mm.yyyy
5	**105**	Italian	5 = dd-mm-yy 105 = dd-mm-yyyy
6	**106** [(1)]	-	6 = dd mon yy 106 = dd mon yyyy
7	**107** [(1)]	-	7 = Mon dd, yy 107 = Mon dd, yyyy
8	**108**	-	hh:mi:ss
-	**9** or **109** [(1,2)]	Default + milliseconds	mon dd yyyy hh:mi:ss:mmmAM (or PM)
10	**110**	USA	10 = mm-dd-yy 110 = mm-dd-yyyy
11	**111**	JAPAN	11 = yy/mm/dd 111 = yyyy/mm/dd
12	**112**	ISO	12 = yymmdd 112 = yyyymmdd
-	**13** or **113** [(1,2)]	Europe default + milliseconds	dd mon yyyy hh:mi:ss:mmm(24h)
14	**114**	-	hh:mi:ss:mmm(24h)
-	**20** or **120** [(2)]	ODBC canonical	yyyy-mm-dd hh:mi:ss(24h)
-	**21** or **121** [(2)]	ODBC canonical (with milliseconds) default for time, date, datetime2, and datetimeoffset	yyyy-mm-dd hh:mi:ss.mmm(24h)
-	**126** [(4)]	ISO8601	yyyy-mm-ddThh:mi:ss.mmm (no spaces) Note: For a milliseconds (mmm) value of 0, the millisecond decimal fraction value

			will not display. For example, the value '2012-11-07T18:26:20.000 displays as '2012-11-07T18:26:20'.
-	**127**[6, 7]	ISO8601 with time zone Z.	yyyy-mm-ddThh:mi:ss.mmmZ (no spaces) Note: For a milliseconds (mmm) value of 0, the millisecond decimal value will not display. For example, the value '2012-11-07T18:26:20.000 will display as '2012-11-07T18:26:20'.
-	**130** [1,2]	Hijri [5]	dd mon yyyy hh:mi:ss:mmmAM In this style, **mon** represents a multi-token Hijri unicode representation of the full month name. This value does not render correctly on a default US installation of SSMS.
-	**131** [2]	Hijri [5]	dd/mm/yyyy hh:mi:ss:mmmAM

[1] These style values return nondeterministic results. Includes all (yy) (without century) styles and a subset of (yyyy) (with century) styles.

[2] The default values (**0** or **100**, **9** or **109**, **13** or **113**, **20** or **120**, and **21** or **121**) always return the century (yyyy).

[3] Input when you convert to **datetime**; output when you convert to character data.

[4] Designed for XML use. For conversion from **datetime** or **smalldatetime** to character data, see the previous table for the output format.

[5] Hijri is a calendar system with several variations. SQL Server uses the Kuwaiti algorithm.

> **Important**
>
> By default, SQL Server interprets two-digit years based on a cutoff year of 2049. That means that SQL Server interprets the two-digit year 49 as 2049 and the two-digit year 50 as 1950. Many client applications, including those based on Automation objects, use a cutoff year of 2030. SQL Server provides the two digit year cutoff configuration option to change the cutoff year used by SQL Server. This allows for the consistent treatment of dates. We recommend specifying four-digit years.

[6] Only supported when casting from character data to **datetime** or **smalldatetime**. When casting character data representing only date or only time components to the **datetime** or **smalldatetime** data types, the unspecified time component is set to 00:00:00.000, and the unspecified date component is set to 1900-01-01.

[7]Use the optional time zone indicator **Z** to make it easier to map XML **datetime** values that have time zone information to SQL Server **datetime** values that have no time zone. Z indicates time zone UTC-0. The HH:MM offset, in the + or - direction, indicates other time zones. For example: `2006-12-12T23:45:12-08:00`.

When converting **smalldatetime** to character data, the styles that include seconds or milliseconds show zeros in these positions. When converting from **datetime** or **smalldatetime** values, use an appropriate **char** or **varchar** data type length to truncate unwanted date parts.

When converting character data to **datetimeoffset**, using a style that includes a time, a time zone offset is appended to the result.

float and real styles

For a **float** or **real** *expression*, *style* can have one of the values shown in the following table. Other values are processed as 0.

Value	Output
0 (default)	A maximum of 6 digits. Use in scientific notation, when appropriate.
1	Always 8 digits. Always use in scientific notation.
2	Always 16 digits. Always use in scientific notation.
3	Always 17 digits. Use for lossless conversion. With this style, every distinct float or real value is guaranteed to convert to a distinct character string. **Applies to:** Azure SQL Database, and starting in SQL Server 2016 (13.x).
126, 128, 129	Included for legacy reasons; a future release could deprecate these values.

money and smallmoney styles

For a **money** or **smallmoney** *expression*, *style* can have one of the values shown in the following table. Other values are processed as 0.

Value	Output
0 (default)	No commas every three digits to the left of the decimal point, and two digits to the right of the decimal point Example: 4235.98.
1	Commas every three digits to the left of the decimal point, and two digits to the right of the decimal point

	Example: 3,510.92.
2	No commas every three digits to the left of the decimal point, and four digits to the right of the decimal point Example: 4235.9819.
126	Equivalent to style 2, when converting to char(n) or varchar(n)

xml styles

For an **xml** *expression*, *style* can have one of the values shown in the following table. Other values are processed as 0.

Value	Output
0 (default)	Use default parsing behavior that discards insignificant white space, and does not allow for an internal DTD subset. **Note:** When converting to the **xml** data type, SQL Server insignificant white space is handled differently than in XML 1.0. For more information, see Create Instances of XML Data.
1	Preserve insignificant white space. This style setting sets the default **xml:space** handling to match the behavior of **xml:space="preserve"**.
2	Enable limited internal DTD subset processing. If enabled, the server can use the following information that is provided in an internal DTD subset, to perform nonvalidating parse operations. - Defaults for attributes are applied - Internal entity references are resolved and expanded - The DTD content model is checked for syntactical correctness The parser ignores external DTD subsets. Also, it does not evaluate the XML declaration to see whether the **standalone** attribute has a **yes** or **no** value. Instead, it parses the XML instance as a stand-alone document.
3	Preserve insignificant white space, and enable limited internal DTD subset processing.

Binary styles

For a **binary(n)**, **char(n)**, **varbinary(n)**, or **varchar(n)** *expression*, *style* can have one of the values shown in the following table. Style values not listed in the table will return an error.

Value	Output

0 (default)	Translates ASCII characters to binary bytes, or binary bytes to ASCII characters. Each character or byte is converted 1:1. For a binary *data_type*, the characters 0x are added to the left of the result.
1, **2**	For a binary *data_type*, the expression must be a character expression. The *expression* must have an **even** number of hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, a, b, c, d, e, f). If the *style* is set to 1, the must have 0x as the first two characters. If the expression contains an odd number of characters, or if any of the characters is invalid, an error is raised. If the length of the converted expression exceeds the length of the *data_type*, the result is right truncated. Fixed length *data_types* larger than the converted result have zeros added to the right of the result. A *data_type* of type character requires a binary expression. Each binary character is converted into two hexadecimal characters. If the length of the converted expression exceeds the length of the *data_type*, it will be right truncated. For a fixed size character type *data_type*, if the length of the converted result is less than its length of the *data_type*, spaces are added to the right of the converted expression, to maintain an even number of hexadecimal digits. The characters 0x will be added to the left of the converted result for *style* 1.

Implicit conversions

Implicit conversions do not require specification of either the CAST function or the CONVERT function. Explicit conversions require specification of the CAST function or the CONVERT function. The following illustration shows all explicit and implicit data type conversions allowed for SQL Server system-supplied data types. These include **bigint**, and **sql_variant**, and **xml**. There is no implicit conversion on assignment from the **sql_variant** data type, but there is implicit conversion to **sql_variant**.

> **Tip**
>
> The Microsoft Download Center has this chart available for download as a PDF file.

When you convert between **datetimeoffset** and the character types **char**, **nchar**, **nvarchar**, and **varchar**, the converted time zone offset part should always have double digits for both HH and MM. For example, -08:00.

> **Note**
>
> Because Unicode data always uses an even number of bytes, use caution when you convert **binary** or **varbinary** to or from Unicode supported data types. For example, the following conversion does not return a hexadecimal value of 41. It returns a hexadecimal value of 4100:
> `SELECT CAST(CAST(0x41 AS nvarchar) AS varbinary)`.

Large-value data types

Large-value data types have the same implicit and explicit conversion behavior as their smaller counterparts - specifically, the **nvarchar**, **varbinary**, and **varchar** data types. However, consider the following guidelines:

- Conversion from **image** to **varbinary(max)**, and vice-versa, operates as an implicit conversion, as do conversions between **text** and **varchar(max)**, and **ntext** and **nvarchar(max)**.
- Conversion from large-value data types, such as **varchar(max)**, to a smaller counterpart data type, such as **varchar**, is an implicit conversion, but truncation occurs if the size of the large value exceeds the specified length of the smaller

data type.

- Conversion from **nvarchar**, **varbinary**, or **varchar** to their corresponding large-value data types happens implicitly.
- Conversion from the **sql_variant** data type to the large-value data types is an explicit conversion.
- Large-value data types cannot be converted to the **sql_variant** data type.

For more information about conversion from the **xml** data type, see Create Instances of XML Data.

xml data type

When you explicitly or implicitly cast the **xml** data type to a string or binary data type, the content of the **xml** data type is serialized based on a defined set of rules. For information about these rules, see Define the Serialization of XML Data. For information about conversion from other data types to the **xml** data type, see Create Instances of XML Data.

text and image data types

The **text** and **image** data types do not support automatic data type conversion. You can explicitly convert **text** data to character data, and **image** data to **binary** or **varbinary**, but the maximum length is 8000 bytes. If you try an incorrect conversion, for example trying to convert a character expression that includes letters to an **int**, SQL Server returns an error message.

Output Collation

When the CAST or CONVERT functions output a character string, and they receive a character string input, the output has the same collation and collation label as the input. If the input is not a character string, the output has the default collation of the database, and a collation label of coercible-default. For more information, see Collation Precedence (Transact-SQL).

To assign a different collation to the output, apply the COLLATE clause to the result expression of the CAST or CONVERT function. For example:

```
SELECT CAST('abc' AS varchar(5)) COLLATE French_CS_AS
```

Truncating and rounding results

When converting character or binary expressions (**binary**, **char**, **nchar**, **nvarchar**, **varbinary**, or **varchar**) to an expression of a different data type, the conversion operation could truncate the output data, only partially display the output data, or return an error. These cases will occur if the result is too short to display. Conversions to **binary**, **char**, **nchar**, **nvarchar**, **varbinary**, or **varchar** are truncated, except for the

to **binary**, **char**, **nchar**, **nvarchar**, **varbinary**, or **varchar** are truncated, except for the conversions shown in the following table.

From data type	To data type	Result
int, **smallint**, or **tinyint**	**char**	*
	varchar	*
	nchar	E
	nvarchar	E
money, **smallmoney**, **numeric**, **decimal**, **float**, or **real**	**char**	E
	varchar	E
	nchar	E
	nvarchar	E

* = Result length too short to display

E = Error returned because result length is too short to display.

SQL Server guarantees that only roundtrip conversions, in other words conversions that convert a data type from its original data type and back again, yield the same values from version to version. The following example shows such a roundtrip conversion:

SQL Copy

```
DECLARE @myval decimal (5, 2);
SET @myval = 193.57;
SELECT CAST(CAST(@myval AS varbinary(20)) AS decimal(10,5));
-- Or, using CONVERT
SELECT CONVERT(decimal(10,5), CONVERT(varbinary(20), @myval));
```

Note

Do not construct **binary** values, and then convert them to a data type of the numeric data type category. SQL Server does not guarantee that the result of a **decimal** or **numeric** data type conversion, to **binary**, will be the same between versions of SQL Server.

The following example shows a resulting expression that is too small to display.

SQL Copy

```
USE AdventureWorks2012;
```

```
GO
SELECT p.FirstName, p.LastName, SUBSTRING(p.Title, 1, 25) AS Title,
    CAST(e.SickLeaveHours AS char(1)) AS [Sick Leave]
FROM HumanResources.Employee e JOIN Person.Person p
    ON e.BusinessEntityID = p.BusinessEntityID
WHERE NOT e.BusinessEntityID >5;
```

Here is the result set.

Copy

```
FirstName   LastName      Title   Sick Leave
---------   ------------- ------- ---------`
Ken         Sanchez       NULL    *
Terri       Duffy         NULL    *
Roberto     Tamburello    NULL    *
Rob         Walters       NULL    *
Gail        Erickson      Ms.     *
(5 row(s) affected)
```

When you convert data types that differ in decimal places, SQL Server will sometimes return a truncated result value, and at other times it will return a rounded value. This table shows the behavior.

From	To	Behavior
numeric	**numeric**	Round
numeric	**int**	Truncate
numeric	**money**	Round
money	**int**	Round
money	**numeric**	Round
float	**int**	Truncate
float	**numeric**	Round Conversion of **float** values that use scientific notation to **decimal** or **numeric** is restricted to values of precision 17 digits only. Any value with precision higher than 17 rounds to zero.
float	**datetime**	Round
datetime	**int**	Round

For example, the values 10.6496 and -10.6496 may be truncated or rounded during conversion to **int** or **numeric** types:

SQL Copy

```
SELECT  CAST(10.6496 AS int) as trunc1,
        CAST(-10.6496 AS int) as trunc2,
        CAST(10.6496 AS numeric) as round1,
        CAST(-10.6496 AS numeric) as round2;
```

Results of the query are shown in the following table:

trunc1	trunc2	round1	round2
10	-10	11	-11

When converting data types where the target data type has fewer decimal places than the source data type, the value is rounded. For example, this conversion returns `$10.3497`:

```
SELECT CAST(10.3496847 AS money);
```

SQL Server returns an error message when converting nonnumeric **char**, **nchar**, **nvarchar**, or **varchar** data to **decimal**, **float**, **int**, **numeric**. SQL Server also returns an error when an empty string (" ") is converted to **numeric** or **decimal**.

Certain datetime conversions are nondeterministic

The following table lists the styles for which the string-to-datetime conversion is nondeterministic.

All styles below 100[1]	106
107	109
113	130

[1] With the exception of styles 20 and 21

Supplementary characters (surrogate pairs)

Starting with SQL Server 2012 (11.x), when using supplementary character (SC) collations, a CAST operation from **nchar** or **nvarchar** to an **nchar** or **nvarchar** type of smaller length will not truncate inside a surrogate pair. Instead, the operation truncates before the supplementary character. For example, the following code fragment leaves `@x` holding just `'ab'`. There is not enough space to hold the supplementary character.

SQL Copy

```
DECLARE @x NVARCHAR(10) = 'ab' + NCHAR(0x10000);
```

```
DECLARE @x NVARCHAR(10) = 'ab' + NCHAR(0x10000);
SELECT CAST (@x AS NVARCHAR(3));
```

When using SC collations, the behavior of `CONVERT`, is analogous to that of `CAST`.

Compatibility support

In earlier versions of SQL Server, the default style for CAST and CONVERT operations on **time** and **datetime2** data types is 121, except when either type is used in a computed column expression. For computed columns, the default style is 0. This behavior impacts computed columns when they are created, used in queries involving auto-parameterization, or used in constraint definitions.

Under compatibility level 110 and higher, the CAST and CONVERT operations on the **time** and **datetime2** datatypes always have 121 as the default style. If a query relies on the old behavior, use a compatibility level less than 110, or explicitly specify the 0 style in the affected query.

Upgrading the database to compatibility level 110 and higher will not change user data that has been stored to disk. You must manually correct this data as appropriate. For example, if you used SELECT INTO to create a table from a source containing a computed column expression described above, the data (using style 0) would be stored rather than the computed column definition itself. You must manually update this data to match style 121.

Examples

A. Using both CAST and CONVERT

These examples retrieve the name of the product, for those products that have a `3` as the first digit of list price, and converts their `ListPrice` values to `int`.

SQL Copy

```
-- Use CAST
USE AdventureWorks2012;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CAST(ListPrice AS int) LIKE '3%';
GO

-- Use CONVERT.
USE AdventureWorks2012;
GO
SELECT SUBSTRING(Name, 1, 30) AS ProductName, ListPrice
FROM Production.Product
WHERE CONVERT(int, ListPrice) LIKE '3%';
GO
```

B. Using CAST with arithmetic operators

This example calculates a single column computation (`Computed`) by dividing the total year-to-date sales (`SalesYTD`) by the commission percentage (`CommissionPCT`). This value is rounded to the nearest whole number and is then CAST to an `int` data type.

SQL Copy

```
USE AdventureWorks2012;
GO
SELECT CAST(ROUND(SalesYTD/CommissionPCT, 0) AS int) AS Computed
FROM Sales.SalesPerson
WHERE CommissionPCT != 0;
GO
```

Here is the result set.

Copy

```
Computed
------
379753754
346698349
257144242
176493899
281101272
0
301872549
212623750
298948202
250784119
239246890
101664220
124511336
97688107
(14 row(s) affected)
```

C. Using CAST to concatenate

This example concatenates noncharacter expressions by using CAST. It uses the AdventureWorksDW database.

SQL Copy

```
SELECT 'The list price is ' + CAST(ListPrice AS varchar(12)) AS ListPrice
FROM dbo.DimProduct
WHERE ListPrice BETWEEN 350.00 AND 400.00;
```

Here is the result set.

Copy

```
ListPrice
-----------------------
The list price is 357.06
The list price is 364.09
The list price is 364.09
The list price is 364.09
The list price is 364.09
```

D. Using CAST to produce more readable text

This example uses CAST in the SELECT list, to convert the `Name` column to a **char(10)** column. It uses the AdventureWorksDW database.

SQL Copy

```
SELECT DISTINCT CAST(EnglishProductName AS char(10)) AS Name, ListPrice
FROM dbo.DimProduct
WHERE EnglishProductName LIKE 'Long-Sleeve Logo Jersey, M';
```

Here is the result set.

Copy

```
Name       ListPrice
---------- ---------
Long-Sleev 31.2437
Long-Sleev 32.4935
Long-Sleev 49.99
```

E. Using CAST with the LIKE clause

This example converts the `money` column `SalesYTD` values to data type `int`, and then to data type `char(20)`, so that the `LIKE` clause can use it.

SQL Copy

```
USE AdventureWorks2012;
GO
SELECT p.FirstName, p.LastName, s.SalesYTD, s.BusinessEntityID
FROM Person.Person AS p
JOIN Sales.SalesPerson AS s
    ON p.BusinessEntityID = s.BusinessEntityID
WHERE CAST(CAST(s.SalesYTD AS int) AS char(20)) LIKE '2%';
GO
```

Here is the result set.

Copy

```
FirstName       LastName            SalesYTD        BusinessEntityID
--------------- ------------------- --------------- -------------
Tsvi            Reiter              2811012.7151    279
```

```
Syed            Abbas           219088.8836     288
Rachel          Valdez          2241204.0424    289
(3 row(s) affected)
```

F. Using CONVERT or CAST with typed XML

These examples show use of CONVERT to convert data to typed XML, by using the XML Data Type and Columns (SQL Server).

This example converts a string with white space, text and markup into typed XML, and removes all insignificant white space (boundary white space between nodes):

SQL Copy

```
SELECT CONVERT(XML, '<root><child/></root>')
```

This example converts a similar string with white space, text and markup into typed XML and preserves insignificant white space (boundary white space between nodes):

SQL Copy

```
SELECT CONVERT(XML, '<root>          <child/>        </root>', 1)
```

This example casts a string with white space, text, and markup into typed XML:

SQL Copy

```
SELECT CAST('<Name><FName>Carol</FName><LName>Elliot</LName></Name>'  AS XML)
```

See Create Instances of XML Data for more examples.

G. Using CAST and CONVERT with datetime data

Starting with GETDATE() values, this example displays the current date and time, uses `CAST` to change the current date and time to a character data type, and then uses `CONVERT` to display the date and time in the `ISO 8601` format.

SQL Copy

```
SELECT
   GETDATE() AS UnconvertedDateTime,
   CAST(GETDATE() AS nvarchar(30)) AS UsingCast,
   CONVERT(nvarchar(30), GETDATE(), 126) AS UsingConvertTo_ISO8601  ;
GO
```

Here is the result set.

Copy

```
```

```
UnconvertedDateTime     UsingCast              UsingConvertTo_ISO8601
----------------------- ---------------------- ----------------------------
2006-04-18 09:58:04.570 Apr 18 2006  9:58AM    2006-04-18T09:58:04.570
(1 row(s) affected)
```

This example is approximately the opposite of the previous example. This example displays a date and time as character data, uses `CAST` to change the character data to the `datetime` data type, and then uses `CONVERT` to change the character data to the `datetime` data type.

SQL Copy

```
SELECT
   '2006-04-25T15:50:59.997' AS UnconvertedText,
   CAST('2006-04-25T15:50:59.997' AS datetime) AS UsingCast,
   CONVERT(datetime, '2006-04-25T15:50:59.997', 126) AS UsingConvertFrom_ISO8
GO
```

Here is the result set.

Copy

```
UnconvertedText         UsingCast              UsingConvertFrom_ISO8601
----------------------- ---------------------- -----------------------
2006-04-25T15:50:59.997 2006-04-25 15:50:59.997 2006-04-25 15:50:59.997
(1 row(s) affected)
```

H. Using CONVERT with binary and character data

These examples show the results of binary and character data conversion, using different styles.

SQL Copy

```
--Convert the binary value 0x4E616d65 to a character value.
SELECT CONVERT(char(8), 0x4E616d65, 0) AS [Style 0, binary to character];
```

Here is the result set.

Copy

```
Style 0, binary to character
----------------------------
Name
(1 row(s) affected)
```

This example shows that Style 1 can force result truncation. The characters 0x in the result set force the truncation.

SQL

```
SELECT CONVERT(char(8), 0x4E616d65, 1) AS [Style 1, binary to character];
```

Here is the result set.

```
Style 1, binary to character
----------------------------
0x4E616D
(1 row(s) affected)
```

This example shows that Style 2 does not truncate the result, because the result does not include the characters 0x.

SQL

```
SELECT CONVERT(char(8), 0x4E616d65, 2) AS [Style 2, binary to character];
```

Here is the result set.

```
Style 2, binary to character
----------------------------
4E616D65
(1 row(s) affected)
```

Convert the character value 'Name' to a binary value.

SQL

```
SELECT CONVERT(binary(8), 'Name', 0) AS [Style 0, character to binary];
```

Here is the result set.

```
Style 0, character to binary
----------------------------
0x4E616D6500000000
(1 row(s) affected)
```

SQL

```
SELECT CONVERT(binary(4), '0x4E616D65', 1) AS [Style 1, character to binary];
```

Here is the result set.

```
Style 1, character to binary
----------------------------
0x4E616D65
(1 row(s) affected)
```

SQL

```sql
SELECT CONVERT(binary(4), '4E616D65', 2) AS [Style 2, character to binary];
```

Here is the result set.

```
Style 2, character to binary
--------------------------------
0x4E616D65
(1 row(s) affected)
```

I. Converting date and time data types

This example shows the conversion of date, time, and datetime data types.

SQL

```sql
DECLARE @d1 date, @t1 time, @dt1 datetime;
SET @d1 = GETDATE();
SET @t1 = GETDATE();
SET @dt1 = GETDATE();
SET @d1 = GETDATE();
-- When converting date to datetime the minutes portion becomes zero.
SELECT @d1 AS [date], CAST (@d1 AS datetime) AS [date as datetime];
-- When converting time to datetime the date portion becomes zero
-- which converts to January 1, 1900.
SELECT @t1 AS [time], CAST (@t1 AS datetime) AS [time as datetime];
-- When converting datetime to date or time non-applicable portion is dropped
SELECT @dt1 AS [datetime], CAST (@dt1 AS date) AS [datetime as date],
   CAST (@dt1 AS time) AS [datetime as time];
```

Examples: Azure SQL Data Warehouse and Parallel Data Warehouse

J. Using CAST and CONVERT

This example retrieves the name of the product for those products that have a `3` in the first digit of their list price, and converts the `ListPrice` of these products to **int**. It uses the AdventureWorksDW database.

SQL Copy

```
SELECT EnglishProductName AS ProductName, ListPrice
FROM dbo.DimProduct
WHERE CAST(ListPrice AS int) LIKE '3%';
```

This example shows the same query, using CONVERT instead of CAST. It uses the AdventureWorksDW database.

SQL Copy

```
SELECT EnglishProductName AS ProductName, ListPrice
FROM dbo.DimProduct
WHERE CONVERT(int, ListPrice) LIKE '3%';
```

K. Using CAST with arithmetic operators

This example calculates a single column value by dividing the product unit price (`UnitPrice`) by the discount percentage (`UnitPriceDiscountPct`). This result is then rounded to the nearest whole number, and finally converted to an `int` data type. This example uses the AdventureWorksDW database.

SQL Copy

```
SELECT ProductKey, UnitPrice,UnitPriceDiscountPct,
       CAST(ROUND (UnitPrice*UnitPriceDiscountPct,0) AS int) AS DiscountPrice
FROM dbo.FactResellerSales
WHERE SalesOrderNumber = 'SO47355'
      AND UnitPriceDiscountPct > .02;
```

Here is the result set.

Copy

```
ProductKey  UnitPrice  UnitPriceDiscountPct  DiscountPrice
----------  ---------  --------------------  -------------
323         430.6445   0.05                  22
213         18.5043    0.05                  1
456         37.4950    0.10                  4
456         37.4950    0.10                  4
216         18.5043    0.05                  1
```

L. Using CAST with the LIKE clause

This example converts the **money** column `ListPrice` to an **int** type, and then to a **char(20)** type, so that the LIKE clause can use it. This example uses the AdventureWorksDW database.

SQL Copy

```
SELECT EnglishProductName AS Name, ListPrice
FROM dbo.DimProduct
WHERE CAST(CAST(ListPrice AS int) AS char(20)) LIKE '2%';
```

M. Using CAST and CONVERT with datetime data

This example displays the current date and time, uses CAST to change the current date and time to a character data type, and finally uses CONVERT display the date and time in the ISO 8601 format. This example uses the AdventureWorksDW database.

SQL Copy

```
SELECT TOP(1)
   SYSDATETIME() AS UnconvertedDateTime,
   CAST(SYSDATETIME() AS nvarchar(30)) AS UsingCast,
   CONVERT(nvarchar(30), SYSDATETIME(), 126) AS UsingConvertTo_ISO8601
FROM dbo.DimCustomer;
```

Here is the result set.

Copy

```
UnconvertedDateTime     UsingCast                   UsingConvertTo_ISO8601
---------------------   -------------------------   ----------------------
07/20/2010 1:44:31 PM   2010-07-20 13:44:31.5879025   2010-07-20T13:44:31.587
```

This example is the rough opposite of the previous example. This example displays a date and time as character data, uses CAST to change the character data to the **datetime** data type, and then uses CONVERT to change the character data to the **datetime** data type. This example uses the AdventureWorksDW database.

SQL Copy

```
SELECT TOP(1)
   '2010-07-25T13:50:38.544' AS UnconvertedText,
CAST('2010-07-25T13:50:38.544' AS datetime) AS UsingCast,
   CONVERT(datetime, '2010-07-25T13:50:38.544', 126) AS UsingConvertFrom_ISO8
FROM dbo.DimCustomer;
```

Here is the result set.

Copy

```
UnconvertedText         UsingCast              UsingConvertFrom_ISO8601
----------------------- ---------------------- ------------------------
2010-07-25T13:50:38.544 07/25/2010 1:50:38 PM   07/25/2010 1:50:38 PM
```

See also

Data Type Conversion (Database Engine)
FORMAT (Transact-SQL)
STR (Transact-SQL)
SELECT (Transact-SQL)
System Functions (Transact-SQL)
Write International Transact-SQL Statements