



HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

SQL SERVER

TS Lê Thị Tú Kiên
kienltt@hnue.edu.vn

Murach's SQL
Server 2012, C8

<http://fit.hnue.edu.vn/~kienltt/SQLSERVER/>

Lecture 2

Data types and functions

(Các kiểu dữ liệu và hàm)

Murach's SQL Server 2012, C8

Lê Thị Tú Kiên - HQT SQL Server

Slide 2

Các kiểu dữ liệu trong SQL Server

Install AP database

Nội dung

- Các kiểu dữ liệu
- Một số hàm chuyển đổi kiểu dữ liệu
- Cách sử dụng các hàm

SQL Server data type categories

(Các kiểu dữ liệu trong SQL Server được chia ra làm 4 nhóm chính)

- String (Xâu kí tự)
- Numeric (Số)
- Temporal (date/time)
- Other (Một số kiểu khác)

Các kiểu dữ liệu trong SQL Server được chia ra làm 4 nhóm chính: xâu kí tự (string), số (numeric), thời gian và khác.

ANSI-standard data types and SQL Server equivalents

(Bảng tương ứng giữa các kiểu dữ liệu trong SQL Server với các kiểu dữ liệu của chuẩn ANSI)

Synonym for ANSI-standard data type	SQL Server data type used
binary varying	varbinary
char varying	varchar
character	character varying
dec	decimal
double precision	float
float	real or float
integer	int

Bảng tương ứng giữa các kiểu dữ liệu trong SQL Server với các kiểu dữ liệu của chuẩn ANSI.

ANSI-standard data types and SQL Server equivalents (continued)

Synonym for ANSI-standard data type	SQL Server data type used
national char	nchar
national char varying	nvarchar
national text	ntext
rowversion	timestamp

The integer data types (Các kiểu DL số nguyên)

Type	Bytes
bigint	8
int	4
smallint	2
tinyint	1
bit	1

Murach's SQL Server 2012, C8

Lê Thị Tú Kiên - HQT SQL Server

Slide 7

Các kiểu dữ liệu số nguyên và số byte dùng để lưu trữ mỗi số trong từng kiểu dữ liệu

The decimal data types (Kiểu dữ liệu số thập phân)

Type	Bytes
decimal[(p,s)]	5-17
numeric[(p,s)]	5-17
money	8
smallmoney	4

Murach's SQL Server 2012, C8

Lê Thị Tú Kiên - HQT SQL Server

Slide 8

Các kiểu số thập phân

The real data types (Kiểu số thực)

Type	Bytes
float[(n)]	4 or 8
real	4

Murach's SQL Server 2012, C8

Lê Thị Tú Kiên - HQT SQL Server

Slide 9

Các kiểu dữ liệu số thực

String data types for storing standard characters

(Kiểu xâu kí tự lưu trữ kí tự chuẩn)

Type	Bytes
char[(n)]	n
varchar[(n)]	

String data types for storing Unicode characters

(Kiểu xâu kí tự lưu trữ kí tự Unicode)

Type	Bytes
nchar(n)	$2 \times n$
nvarchar(n)	

Các kiểu dữ liệu xâu kí tự lưu trữ kí tự chuẩn và các kiểu dữ liệu xâu kí tự (bộ mã chuẩn) lưu trữ các kí tự Unicode (bộ mã mở rộng).

Date/time data types prior to SQL Server 2008

(Kiểu dữ liệu ngày tháng trong các phiên bản SQL Server trước năm 2008)

Type	Bytes
datetime	8
smalldatetime	4

Date/time data types for SQL Server 2008 and later

(Kiểu dữ liệu ngày tháng trong các phiên bản SQL Server từ năm 2008 trở lại đây)

Type	Bytes
date	3
time(n)	3-5
datetime2(n)	6-8
datetimeoffset(n)	8-10

Các kiểu dữ liệu ngày tháng

Common date formats

Format	Example
yyyy-mm-dd	2012-04-30
mm/dd/yyyy	4/30/2012
mm-dd-yy	4-30-12
Month dd, yyyy	April 30, 2012
Mon dd, yy	Apr 30, 12
dd Mon yy	30 Apr 12

Common time formats

Format	Example
hh:mi	16:20
hh:mi am/pm	4:20 pm
hh:mi:ss	4:20:36
hh:mi:ss:mmm	4:20:36:12
hh:mi:ss.nnnnnnnn	4:20:36.1234567

Các định dạng cho kiểu dữ liệu ngày tháng

Một số chú ý khi sử dụng kiểu dữ liệu ngày tháng

- Giá trị một ngày tháng đặt trong cặp ngoặc đơn.
- Nếu NSD (người sử dụng) không xác định thời gian trong một giá trị kiểu ngày tháng thì thời gian mặc định được lấy là 12:00 a.m.
- Nếu NSD (người sử dụng) không xác định ngày trong một giá trị kiểu ngày tháng thì ngày mặc định được lấy là 1/1/1900.
- Mặc định, với giá trị kiểu năm được lưu dưới dạng hai chữ số thì các năm từ 00 đến 99 được hiểu là các năm từ 2000 đến 2049, còn các năm từ 50 đến 99 được hệ thống hiểu là các năm từ 1950 đến 1999.
- NSD có thể thiết lập thời gian đồng hồ 12 giờ hoặc 24 giờ, mặc định là 12 giờ.

Một số chú ý khi sử dụng kiểu dữ liệu ngày tháng:

- Giá trị một ngày tháng đặt trong cặp ngoặc đơn.
- Nếu NSD (người sử dụng) không xác định thời gian trong một giá trị kiểu ngày tháng thì thời gian mặc định được lấy là 12:00 a.m.
- Nếu NSD (người sử dụng) không xác định ngày trong một giá trị kiểu ngày tháng thì ngày mặc định được lấy là 1/1/1900.
- Mặc định, với giá trị kiểu năm được lưu dưới dạng hai chữ số thì các năm từ 00 đến 99 được hiểu là các năm từ 2000 đến 2049, còn các năm từ 50 đến 99 được hệ thống hiểu là các năm từ 1950 đến 1999.
- NSD có thể thiết lập thời gian đồng hồ 12 giờ hoặc 24 giờ, mặc định là 12 giờ.

How to use date/time literals

To code a date/time literal, enclose the date/time value in single quotes.

If you don't specify a time in a date/time value, the time defaults to 12:00 a.m.

If you don't specify a date in a date/time value, the date defaults to January 1, 1900.

By default, the years 00 to 49 are interpreted as 2000 to 2049 and the years 50

through 99 are interpreted as 1950 through 1999.

You can specify a time using either a 12-hour or a 24-hour clock.

For a 12-hour clock, am is the default.

The large value data types for SQL Server 2005 and later

- varchar(max)
- nvarchar(max)
- varbinary(max)

How the large value data types map to the old large object types

SQL Server 2005 and later	Prior to 2005
varchar(max)	text
nvarchar(max)	ntext
varbinary(max)	image

Các kiểu dữ liệu lớn từ phiên bản SQL Server 2005 đến nay.

Order of precedence for common data types

Precedence	Category	Data type
Highest	Date/time	datetime smalldatetime
	Numeric	float real decimal money smallmoney int smallint tinyint bit nvarchar nchar varchar char
Lowest	String	

- ❖ Các kiểu dữ liệu có thể được tự động chuyển đổi (chuyển đổi không tường minh).
- ❖ Thông thường các kiểu dữ liệu có mức ưu tiên thấp hơn sẽ tự động được chuyển sang kiểu dữ liệu có mức ưu tiên cao hơn.

Thứ tự ưu tiên các kiểu dữ liệu.

Expressions that use implicit conversion

(Ví dụ chuyển đổi kiểu tự động)

```
InvoiceTotal * .0775
    -- InvoiceTotal (money) converted to decimal
PaymentTotal - 100
    -- Numeric literal converted to money
PaymentDate = '2012-04-05'
    -- Date literal converted to smalldatetime value
```

Các kiểu dữ liệu có thể được tự động chuyển đổi (chuyển đổi không tường minh, tự động). Thông thường các kiểu dữ liệu có mức ưu tiên thấp hơn sẽ tự động được chuyển sang kiểu dữ liệu có mức ưu tiên cao hơn. Để chuyển đổi ngược lại (ép kiểu) ta cần dùng các hàm chuyển đổi kiểu dữ liệu một cách tường minh.

Conversions that can't be done implicitly

(Các trường hợp không tự động chuyển đổi)

From data type (Từ kiểu DL)	To data type (Đến kiểu DL)
char, varchar, nchar, nvarchar datetime, smalldatetime	money, smallmoney decimal, numeric, float, real, bigint, int, smallint, tinyint, money, smallmoney, bit
money, smallmoney	char, varchar, nchar, nvarchar

- Để chuyển đổi được các trường hợp trên, ta cần dùng các hàm chuyển đổi kiểu dữ liệu một cách tường minh (ép kiểu).

Các kiểu dữ liệu có thể được tự động chuyển đổi (chuyển đổi không tường minh, tự động). Thông thường các kiểu dữ liệu có mức ưu tiên thấp hơn sẽ tự động được chuyển sang kiểu dữ liệu có mức ưu tiên cao hơn. Để chuyển đổi ngược lại (ép kiểu) ta cần dùng các hàm chuyển đổi kiểu dữ liệu một cách tường minh.

The syntax of the CAST function

(Hàm chuyển đổi kiểu dữ liệu CAST)

`CAST(expression AS data_type)`

A SELECT statement that uses the CAST function

(Ví dụ sử dụng hàm CAST để chuyển đổi kiểu dữ liệu)

```
SELECT InvoiceDate, InvoiceTotal,  
       CAST(InvoiceDate AS varchar) AS varcharDate,  
       CAST(InvoiceTotal AS int) AS integerTotal,  
       CAST(InvoiceTotal AS varchar) AS varcharTotal  
FROM Invoices;
```

	InvoiceDate	InvoiceTotal	varcharDate	integerTotal	varcharTotal
1	2011-12-08 00:00:00	3813.33	Dec 8 2011 12:00AM	3813	3813.33
2	2011-12-10 00:00:00	40.20	Dec 10 2011 12:00AM	40	40.20
3	2011-12-13 00:00:00	138.75	Dec 13 2011 12:00AM	139	138.75
4	2011-12-16 00:00:00	144.70	Dec 16 2011 12:00AM	145	144.70

Hàm chuyển đổi kiểu dữ liệu CAST và ví dụ.

How to convert data when performing integer division

(Cách chuyển kiểu dữ liệu khi thực hiện phép chia kiểu số nguyên)

Operation	Result
50/100	0
50/CAST(100 AS decimal(3))	.500000

Cách chuyển kiểu dữ liệu khi thực hiện phép chia kiểu số nguyên.

The syntax of the CONVERT function

(Hàm chuyển kiểu dữ liệu CONVERT)

```
CONVERT(data_type, expression [, style])
```

Convert and format dates

(Ví dụ sử dụng hàm convert và định dạng ngày tháng)

```
SELECT CONVERT(varchar, InvoiceDate) AS varcharDate,  
       CONVERT(varchar, InvoiceDate, 1) AS varcharDate_1,  
       CONVERT(varchar, InvoiceDate, 107) AS varcharDate_107,  
       CONVERT(varchar, InvoiceTotal) AS varcharTotal,  
       CONVERT(varchar, InvoiceTotal, 1) AS varcharTotal_1  
  FROM Invoices;
```

	varcharDate	varcharDate_1	varcharDate_107	varcharTotal	varcharTotal_1
1	Dec 8 2011 12:00AM	12/08/11	Dec 08, 2011	3813.33	3,813.33
2	Dec 10 2011 12:00AM	12/10/11	Dec 10, 2011	40.20	40.20
3	Dec 13 2011 12:00AM	12/13/11	Dec 13, 2011	138.75	138.75
4	Dec 16 2011 12:00AM	12/16/11	Dec 16, 2011	144.70	144.70

Hàm chuyển kiểu dữ liệu CONVERT và ví dụ

Style codes for converting date/time data to character data

(Mã code để chuyển dữ liệu kiểu ngày tháng sang kiểu kí tự)

Code	Output format
0 or 100 (default)	Mon dd yyyy hh:miAM/PM
1 or 101	mm/dd/yy or mm/dd/yyyy
7 or 107	Mon dd, yy or Mon dd, yyyy
8 or 108	hh:mi:ss
10 or 110	mm-dd-yy or mm-dd-yyyy
12 or 112	yymmdd or yyymmdd
14 or 114	hh:mi:ss:mmm (24-hour clock)

Mã code để chuyển dữ liệu kiểu ngày tháng sang kiểu kí tự.

Style codes for converting real data to character data

(Mã code để chuyển dữ liệu kiểu số thực sang kiểu kí tự)

Code	Output
0 (default)	6 digits maximum
1	8 digits; must use scientific notation
2	16 digits; must use scientific notation

Mã code để chuyển dữ liệu kiểu số thực sang kiểu kí tự.

Style codes for converting money data to character data

(Mã code để chuyển dữ liệu kiểu tiền tệ sang kiểu kí tự)

Code	Output
0 (default)	2 digits to the right of the decimal point; no commas to the left
1	2 digits to the right of the decimal point; commas to the left
2	4 digits to the right of the decimal point; no commas to the left

Mã code để chuyển dữ liệu kiểu tiền tệ sang kiểu kí tự.

The syntax of the TRY_CONVERT function

(Hàm chuyển kiểu dữ liệu TRY_CONVERT)

```
TRY_CONVERT(data_type, expression [, style ])
```

Convert and format dates (Ví dụ)

```
SELECT TRY_CONVERT(varchar, InvoiceDate) AS varcharDate,
       TRY_CONVERT(varchar, InvoiceDate, 1) AS varcharDate_1,
       TRY_CONVERT(varchar, InvoiceDate, 107)
             AS varcharDate_107,
       TRY_CONVERT(varchar, InvoiceTotal) AS varcharTotal,
       TRY_CONVERT(varchar, InvoiceTotal, 1)
             AS varcharTotal_1,
       TRY_CONVERT(date, 'Feb 29 2011') AS invalidDate
  FROM Invoices;
```

	varcharDate	varcharDate_1	varcharDate_107	varcharTotal	varcharTotal_1	invalidDate
1	Dec 8 2011 12:00AM	Dec 8 2011 12:00AM	Dec 8 2011 12:00AM	3813.33	3813.33	NULL
2	Dec 10 2011 12:00AM	Dec 10 2011 12:00AM	Dec 10 2011 12:00AM	40.20	40.20	NULL
3	Dec 13 2011 12:00AM	Dec 13 2011 12:00AM	Dec 13 2011 12:00AM	138.75	138.75	NULL
4	Dec 16 2011 12:00AM	Dec 16 2011 12:00AM	Dec 16 2011 12:00AM	144.70	144.70	NULL
5	Dec 16 2011 12:00AM	Dec 16 2011 12:00AM	Dec 16 2011 12:00AM	15.50	15.50	NULL
6	Dec 16 2011 12:00AM	Dec 16 2011 12:00AM	Dec 16 2011 12:00AM	42.75	42.75	NULL
7	Dec 21 2011 12:00AM	Dec 21 2011 12:00AM	Dec 21 2011 12:00AM	172.50	172.50	NULL
8	Dec 24 2011 12:00AM	Dec 24 2011 12:00AM	Dec 24 2011 12:00AM	95.00	95.00	NULL

Murach's SQL Server 2012, C8

Lê Thị Tú Kiên - HQT SQL Server

Slide 24

Hàm chuyển kiểu dữ liệu TRY_CONVERT và ví dụ

Other data conversion functions

(Một số hàm chuyển kiểu dữ liệu khác)

- **STR(float[,length[,decimal]])**
- **CHAR(integer)**
- **ASCII(string)**
- **NCHAR(integer)**
- **UNICODE(string)**

Examples that use the data conversion functions

(Ví dụ về các hàm chuyển kiểu ở trên)

Function	Result
STR(1234.5678, 7, 1)	1234.6
CHAR(79)	o
ASCII('Orange')	79
NCHAR(332)	o
UNICODE(N'Or')	332

Một số hàm chuyển kiểu dữ liệu khác.

ASCII codes for common control characters

(Mã ASCII cho các kí tự điều khiển)

Control character	Value
Tab	Char(9)
Line feed	Char(10)
Carriage return	Char(13)

Use the CHAR function to format output

(Sử dụng hàm Char để định dạng kết quả)

```
SELECT VendorName + CHAR(13) + CHAR(10)
      + VendorAddress1 + CHAR(13) + CHAR(10)
      + VendorCity + ', ' + VendorState + ' '
  VendorZipCode
FROM Vendors
WHERE VendorID = 1;
```

```
US Postal Service
Attn: Supt. Window Services
Madison, WI 53707
```

Bài tập về nhà

- Thực hiện lại các ví dụ chuyển đổi kiểu dữ liệu tường minh trên CSDL AP-K67A

Cách sử dụng các hàm

(Sinh viên tự tìm hiểu và thực hiện các ví dụ ở nhà)

Some of the string functions

```
LEN(string)
LTRIM(string)
RTRIM(string)
LEFT(string,length)
RIGHT(string,length)
SUBSTRING(string,start,length)
REPLACE(search,find,replace)
REVERSE(string)
CHARINDEX(find,search[,start])
PATINDEX(find,search)
CONCAT(value1,value2[,value3]...)
LOWER(string)
UPPER(string)
SPACE(integer)
```

String function examples

Function	Result
LEN('SQL Server')	10
LEN(' SQL Server ')	12
LEFT('SQL Server', 3)	'SQL'
LTRIM(' SQL Server ')	'SQL Server '
RTRIM(' SQL Server ')	' SQL Server'
LTRIM(RTRIM(' SQL Server '))	'SQL Server'
LOWER('SQL Server')	'sql server'
UPPER('ca')	CA
PATINDEX('%v_r%', 'SQL Server')	8
CHARINDEX('SQL', ' SQL Server')	3
CHARINDEX('-', '(559) 555-1212')	10
SUBSTRING('(559) 555-1212', 7, 8)	555-1212
REPLACE(RIGHT('(559) 555-1212', 13), ' ', '-')	559-555-1212
CONCAT('Run time: ', 1.52, ' seconds')	Run time: 1.52 seconds

A SELECT statement that uses three functions

```
Select VendorName, VendorContactLName + ', '
      + LEFT(VendorContactFName, 1)
      + '.' AS ContactName, RIGHT(VendorPhone, 8) AS Phone
FROM Vendors
WHERE SUBSTRING(VendorPhone, 2, 3) = 559
ORDER BY VendorName;
```

	VendorName	ContactName	Phone
1	Abbey Office Furnishings	Francis, K.	555-8300
2	BFI Industries	Kaleigh, E.	555-1551
3	Bill Marvin Electric Inc	Hostley, K.	555-5106
4	Cal State Termite	Hunter, D.	555-1534
5	California Business Machines	Rohansen, A.	555-5570

How to sort by a string column that contains numbers

Sorted by the ID column

```
SELECT * FROM StringSample  
ORDER BY ID;
```

ID	Name	AltID
1	Lizbeth Darien	01
2	Lance Pinos-Potter	17
3	Damell O'Sullivan	02
4	Jean Paul Renard	20
5	Alisha von Strumpf	03

How to sort by a string column that contains numbers (continued)

Sorted by the ID column cast to an integer

```
SELECT * FROM StringSample  
ORDER BY CAST(ID AS int);
```

ID	Name	AID
1	Lizbeth Darien	01
2	Damell O'Sullivan	02
3	Alisha von Strumpf	03
4	Lance Pinos-Potter	17
5	Jean Paul Renard	20

How to use the string functions to parse a string

```
SELECT Name,  
       LEFT(Name, CHARINDEX(' ', Name) - 1) AS First,  
       RIGHT(Name, LEN(Name) - CHARINDEX(' ', Name) ) AS Last  
FROM StringSample;
```

	Name	First	Last
1	Lizbeth Darien	Lizbeth	Darien
2	Damell O'Sullivan	Damell	O'Sullivan
3	Lance Pinos-Potter	Lance	Pinos-Potter
4	Jean Paul Renard	Jean	Paul Renard
5	Alisha von Strump	Alisha	von Strump

Some of the numeric functions

```
ROUND (number,length[,function])
ISNUMERIC(expression)
ABS (number)
CEILING (number)
FLOOR (number)
SQUARE (float_number)
SQRT (float_number)
RAND ([integer])
```

Examples that use the numeric functions

Function	Result
ROUND(12.5, 0)	13.0
ROUND(12.4999, 0)	12.0000
ROUND(12.4999, 1)	12.5000
ROUND(12.4999, -1)	10.0000
ROUND(12.5, 0, 1)	12.0
ISNUMERIC(-1.25)	1
ISNUMERIC('SQL Server')	0
ISNUMERIC('2012-09-30')	0

Examples that use the numeric functions (cont.)

Function	Result
ABS(-1.25)	1.25
CEILING(-1.25)	-1
FLOOR(-1.25)	-2
CEILING(1.25)	2
FLOOR(1.25)	1
SQUARE(5.2786)	27.86361796
SQRT(125.43)	11.199553562531
RAND()	0.243729

The RealSample table

ID	R
1	1.0000000000000011
2	1
3	0.9999999999999999
4	1234.56789012345
5	999.04440209348
6	24.04849

How to search for approximate real values

A SELECT statement that searches for a range of values

```
SELECT * FROM RealSample  
WHERE R BETWEEN 0.99 AND 1.01;
```

A SELECT statement that searches for rounded values

```
SELECT * FROM RealSample  
WHERE ROUND(R, 2) = 1;
```

ID	R
1	1
2	1
3	0.9999999999999999

A SELECT statement that formats real numbers

```
SELECT ID, R, CAST(R AS decimal(9,3)) AS R_decimal,
       CAST(CAST(R AS decimal(9,3)) AS varchar(9))
          AS R_varchar,
       LEN(CAST(CAST(R AS decimal(9,3)) AS varchar(9)))
          AS R_LEN,
       SPACE(9 - LEN(CAST(CAST(R AS decimal(9,3)) AS varchar(9))))
          AS varchar(9)) +
       CAST(CAST(R AS decimal(9,3)) AS varchar(9))
          AS R_Formatted
FROM RealSample;
```

R_decimal	R_varchar	R_LEN	R_Formatted
1.000	1.000	5	1.000
1.000	1.000	5	1.000
1.000	1.000	5	1.000
1234.568	1234.568	8	1234.568
999.044	999.044	7	999.044
24.048	24.048	6	24.048

Some of the date/time functions

```
GETDATE ()  
GETUTCDATE ()  
  
SYSDATETIME ()  
SYSUTCDATETIME ()  
SYSDATETIMEOFFSET ()  
  
DAY (date)  
MONTH (date)  
YEAR (date)  
DATENAME (datepart,date)  
DATEPART (datepart,date)
```

Some of the date/time functions (continued)

```
DATEADD (datepart, number, date)  
DATEDIFF (datepart, startdate, enddate)  
  
TODATETIMEOFFSET (datetime2, tzoffset)  
SWITCHOFFSET (datetimeoffset, tzoffset)  
  
EOMONTH (startdate [,months])  
DATEFROMPARTS (year, month, day)  
ISDATE (expression)
```

Date part values and abbreviations

Argument	Abbreviations
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw

Date part values and abbreviations (continued)

Argument	Abbreviations
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns
tzoffset	tz

Examples that use date/time functions

Function	Result
GETDATE()	2012-09-30 14:10:13.813
GETUTCDATE()	2012-09-30 21:10:13.813
SYSDATETIME()	2012-09-30 14:10:13.8160822
SYSUTCDATETIME()	2012-09-30 21:10:13.8160822
SYSDATETIMEOFFSET()	2012-09-30 14:10:13.8160822 -07.00
MONTH('2012-09-30')	9
DATEPART(month,'2012-09-30')	9
DATENAME(month,'2012-09-30')	September
DATENAME(m,'2012-09-30')	September

Examples that use date/time functions (continued)

Function	Result
EOMONTH ('2012-02-01')	2012-02-29
EOMONTH ('2012-02-01',2)	2012-04-30
DATEFROMPARTS (2012,4,3)	2012-04-03
ISDATE ('2012-09-30')	1
ISDATE ('2012-09-31')	0
ISDATE ('23:59:59')	1
ISDATE ('23:99:99')	0

Examples that use the DAY, MONTH, and YEAR functions

Function	Result
DAY('2012-09-30')	30
MONTH('2012-09-30')	9
YEAR('2012-09-30')	2012

Examples that use the DATEPART function

Function	Result
DATEPART(day, '2012-09-30 11:35:00')	30
DATEPART(month, '2012-09-30 11:35:00')	9
DATEPART(year, '2012-09-30 11:35:00')	2012
DATEPART(hour, '2012-09-30 11:35:00')	11
DATEPART(minute, '2012-09-30 11:35:00')	35
DATEPART(second, '2012-09-30 11:35:00')	0
DATEPART(quarter, '2012-09-30 11:35:00')	3
DATEPART(dayofyear, '2012-09-30 11:35:00')	273
DATEPART(week, '2012-09-30 11:35:00')	40
DATEPART(weekday, '2012-09-30 11:35:00')	1
DATEPART(millisecond, '11:35:00.1234567')	123
DATEPART(microsecond, '11:35:00.1234567')	123456
DATEPART(nanosecond, '11:35:00.1234567')	123456700
DATEPART(tzoffset, '11:35:00.1234567 -07:00')	-420

Examples that use the DATENAME function

Function	Result
DATENAME(day, '2012-09-30 11:35:00')	30
DATENAME(month, '2012-09-30 11:35:00')	September
DATENAME(year, '2012-09-30 11:35:00')	2012
DATENAME(hour, '2012-09-30 11:35:00')	11
DATENAME(minute, '2012-09-30 11:35:00')	35
DATENAME(second, '2012-09-30 11:35:00')	0
DATENAME(quarter, '2012-09-30 11:35:00')	3
DATENAME(dayofyear, '2012-09-30 11:35:00')	274
DATENAME(week, '2012-09-30 11:35:00')	40
DATENAME(weekday, '2012-09-30 11:35:00')	Sunday
DATENAME(millisecond, '11:35:00.1234567')	123
DATENAME(microsecond, '11:35:00.1234567')	123456
DATENAME(nanosecond, '11:35:00.1234567')	123456700
DATENAME(tzoffset, '11:35:00.1234567 -07:00')	-07:00

Examples that use the DATEADD function

Function	Result
DATEADD(day, 1, '2012-09-30 11:35:00')	2012-10-01 11:35:00.000
DATEADD(month, 1, '2012-09-30 11:35:00')	2012-10-30 11:35:00.000
DATEADD(year, 1, '2012-09-30 11:35:00')	2013-09-30 11:35:00.000
DATEADD(hour, 1, '2012-09-30 11:35:00')	2012-09-30 12:35:00.000
DATEADD(minute, 1, '2012-09-30 11:35:00')	2012-09-30 11:36:00.000
DATEADD(second, 1, '2012-09-30 11:35:00')	2012-09-30 11:35:01.000
DATEADD(quarter, 1, '2012-09-30 11:35:00')	2012-12-30 11:35:00.000
DATEADD(week, 1, '2012-09-30 11:35:00')	2012-10-07 11:35:00.000
DATEADD(month, -1, '2012-09-30 11:35:00')	2012-08-30 11:35:00.000
DATEADD(year, 1.5, '2012-09-30 11:35:00')	2013-09-30 11:35:00.000

Examples that use the DATEDIFF function

Function	Result
DATEDIFF(day, '2011-12-01', '2012-09-30')	304
DATEDIFF(month, '2011-12-01', '2012-09-30')	9
DATEDIFF(year, '2011-12-01', '2012-09-30')	1
DATEDIFF(hour, '06:46:45', '11:35:00')	5
DATEDIFF(minute, '06:46:45', '11:35:00')	289
DATEDIFF(second, '06:46:45', '11:35:00')	17295
DATEDIFF(quarter, '2011-12-01', '2012-09-30')	3
DATEDIFF(week, '2011-12-01', '2012-09-30')	44
DATEDIFF(day, '2012-09-30', '2011-12-01')	-304

Examples that use the addition and subtraction operators

Operation	Result
<code>CAST('2012-09-30 11:35:00' AS smalldatetime) + 1</code>	<code>2012-10-01 11:35:00</code>
<code>CAST('2012-09-30 11:35:00' AS smalldatetime) - 1</code>	<code>2012-09-29 11:35:00</code>
<code>CAST(CAST('2012-09-30' AS datetime) - CAST('2011-12-01' AS datetime) AS int)</code>	<code>304</code>

The contents of the DateSample table

ID	StartDate
1	1982-11-01 00:00:00.000
2	2002-10-28 00:00:00.000
3	2007-06-30 00:00:00.000
4	2008-10-28 10:00:00.000
5	2011-10-28 13:58:32.823
6	2011-11-01 09:02:25.000

A search condition that fails to return a row

```
SELECT * FROM DateSample  
WHERE StartDate = '2011-10-28';
```

SELECT statements that ignore time values

**Use the date type to remove time values
(SQL Server 2008 or later)**

```
SELECT * FROM DateSample  
WHERE CONVERT(date, StartDate) = '2011-10-28';
```

Search for a range of dates

```
SELECT * FROM DateSample  
WHERE StartDate >= '2011-10-28' AND  
      StartDate < '2011-10-29';
```

Search for month, day, and year components

```
SELECT * FROM DateSample  
WHERE MONTH(StartDate) = 10 AND  
      DAY(StartDate) = 28 AND  
      YEAR(StartDate) = 2011;
```

SELECT statements that ignore time values (continued)

Use the CAST function to remove time values

```
SELECT * FROM DateSample  
WHERE CAST(CAST(StartDate AS char(11)) AS datetime)  
= '2011-10-28';
```

Use the CONVERT function to remove time values

```
SELECT * FROM DateSample  
WHERE CONVERT(datetime, CONVERT(char(10), StartDate,  
110)) = '2011-10-28';
```

The result set

ID	StartDate
1	2011-10-28 13:58:32.823

The contents of the DateSample table

ID	StartDate
1	1982-11-01 00:00:00.000
2	2002-10-28 00:00:00.000
3	2007-06-30 00:00:00.000
4	2008-10-28 10:00:00.000
5	2011-10-28 13:58:32.823
6	2011-11-01 09:02:25.000

Two SELECT statements that ignore date values

Use the time type to remove date values
(SQL Server 2008 or later)

```
SELECT * FROM DateSample
WHERE CONVERT(time, StartDate) >= '09:00:00' AND
      CONVERT(time, StartDate) < '12:59:59.999';
```

Use the CONVERT function to remove date values
(prior to SQL Server 2008)

```
SELECT * FROM DateSample
WHERE CONVERT(datetime, CONVERT(char(12), StartDate, 8))
      >= '09:00:00' AND
      CONVERT(datetime, CONVERT(char(12), StartDate, 8))
      < '12:59:59.999';
```

The result set

ID	StartDate
1	2008-10-28 10:00:00.000
2	2011-11-01 09:02:25.000

The syntax of the simple CASE function

```
CASE input_expression
    WHEN when_expression_1 THEN result_expression_1
    [WHEN when_expression_2 THEN result_expression_2]...
    [ELSE else_result_expression]
END
```

A SELECT statement with a simple CASE function

```
SELECT InvoiceNumber, TermsID,
CASE TermsID
    WHEN 1 THEN 'Net due 10 days'
    WHEN 2 THEN 'Net due 20 days'
    WHEN 3 THEN 'Net due 30 days'
    WHEN 4 THEN 'Net due 60 days'
    WHEN 5 THEN 'Net due 90 days'
END AS Terms
FROM Invoices;
```

InvoiceNumber	TermsID	Terms
6	3	Net due 30 days
7	3	Net due 30 days
8	1	Net due 10 days

The syntax of the searched CASE function

```
CASE
    WHEN conditional_expression_1 THEN result_expression_1
    [WHEN conditional_expression_2
        THEN result_expression_2]...
    [ELSE else_result_expression]
END
```

A SELECT statement with a searched CASE function

```
SELECT InvoiceNumber, InvoiceTotal, InvoiceDate,
InvoiceDueDate,
CASE
    WHEN DATEDIFF(day, InvoiceDueDate, GETDATE()) > 30
        THEN 'Over 30 days past due'
    WHEN DATEDIFF(day, InvoiceDueDate, GETDATE()) > 0
        THEN '1 to 30 days past due'
    ELSE 'Current'
END AS Status
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0;
```

The result set

	InvoiceNumber	InvoiceTotal	InvoiceDate	InvoiceDueDate	Status
9	134116	90.36	2012-03-28 00:00:00	2012-04-17 00:00:00	Over 30 days past due
10	0-2436	10976.06	2012-03-31 00:00:00	2012-04-30 00:00:00	1 to 30 days past due
11	547480102	224.00	2012-04-01 00:00:00	2012-04-30 00:00:00	1 to 30 days past due

The syntax of the IIF function

```
IIF(conditional_expression, true_value, false_value)
```

A SELECT statement with an IIF function

```
SELECT VendorID, SUM(InvoiceTotal) AS SumInvoices,  
       IIF(SUM(InvoiceTotal) < 1000, 'Low', 'High')  
             AS InvoiceRange  
FROM Invoices  
GROUP BY VendorID;
```

	VendorID	SumInvoices	InvoiceRange
1	34	1200.12	High
2	37	564.00	Low
3	48	856.92	Low
4	72	21927.31	High
5	80	265.36	Low
6	81	936.93	Low
7	82	600.00	Low
8	83	2154.42	High

The syntax of the CHOOSE function

```
CHOOSE(index, value1, value2 [,value3]...)
```

A SELECT statement with a CHOOSE function

```
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,  
    CHOOSE(TermsID, '10 days', '20 days', '30 days',  
        '60 days', '90 days') AS NetDue  
FROM Invoices  
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0;
```

	InvoiceNumber	InvoiceDate	InvoiceTotal	NetDue
1	39104	2012-03-10 00:00:00	85.31	30 days
2	963253264	2012-03-18 00:00:00	52.25	30 days
3	31361833	2012-03-21 00:00:00	579.42	20 days
4	263253268	2012-03-21 00:00:00	59.97	30 days
5	263253270	2012-03-22 00:00:00	67.92	30 days

The syntax of the COALESCE function

```
COALESCE(expression_1 [, expression_2]...)
```

The syntax of the ISNULL function

```
ISNULL(check_expression, replacement_value)
```

A SELECT statement with a COALESCE function

```
SELECT PaymentDate,  
       COALESCE(PaymentDate, '1900-01-01') AS NewDate  
  FROM Invoices;
```

The same statement with an ISNULL function

```
SELECT PaymentDate,  
       ISNULL(PaymentDate, '1900-01-01') AS NewDate  
  FROM Invoices;
```

The result set

	PaymentDate	NewDate
111	2012-05-03 00:00:00	2012-05-03 00:00:00
112	NULL	1900-01-01 00:00:00
113	NULL	1900-01-01 00:00:00
114	2012-05-04 00:00:00	2012-05-04 00:00:00

A SELECT statement that substitutes a different data type

```
SELECT VendorName,
       COALESCE(CAST(InvoiceTotal AS varchar), 'No invoices')
               AS InvoiceTotal
  FROM Vendors LEFT JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
 ORDER BY VendorName;
```

	VendorName	InvoiceTotal
1	Abbey Office Furnishings	17.50
2	American Booksellers Assoc	No invoices
3	American Express	No invoices
4	ASC Signs	No invoices
5	Ascom Hasler Mailing Systems	No invoices

The syntax of the GROUPING function

```
GROUPING(column_name)
```

A summary query with a GROUPING function

```
SELECT
    CASE
        WHEN GROUPING(VendorState) = 1 THEN 'All'
        ELSE VendorState
    END AS VendorState,
    CASE
        WHEN GROUPING(VendorCity) = 1 THEN 'All'
        ELSE VendorCity
    END AS VendorCity,
    COUNT(*) AS QtyVendors
FROM Vendors
WHERE VendorState IN ('IA', 'NJ')
GROUP BY VendorState, VendorCity WITH ROLLUP
ORDER BY VendorState DESC, VendorCity DESC;
```

The result set

	VendorState	VendorCity	QtyVendors
1	NJ	Washington	1
2	NJ	Fairfield	1
3	NJ	East Brunswick	2
4	NJ	All	4
5	IA	Washington	1
6	IA	Fairfield	1
7	IA	All	2
8	All	All	6

The syntax for the four ranking functions

```
ROW_NUMBER()  
    OVER ([partition_by_clause] order_by_clause)  
  
RANK()  
    OVER ([partition_by_clause] order_by_clause)  
  
DENSE_RANK()  
    OVER ([partition_by_clause] order_by_clause)  
  
NTILE(integer_expression)  
    OVER ([partition_by_clause] order_by_clause)
```

A query with a ROW_NUMBER function

```
SELECT ROW_NUMBER() OVER(ORDER BY VendorName) AS RowNumber,  
      VendorName  
   FROM Vendors;
```

	RowNumber	VendorName
1	1	Abbey Office Furnishings
2	2	American Booksellers Assoc
3	3	American Express
4	4	ASC Signs
5	5	Ascom Hasler Mailing Systems

A query that uses the PARTITION BY clause

```
SELECT ROW_NUMBER() OVER(PARTITION BY VendorState  
ORDER BY VendorName) As RowNumber, VendorName,  
VendorState  
FROM Vendors;
```

	RowNumber	VendorName	VendorState
1	1	AT&T	AZ
2	2	Computer Library	AZ
3	3	Wells Fargo Bank	AZ
4	1	Abbey Office Furnishings	CA
5	2	American Express	CA
6	3	ASC Signs	CA

A query with RANK and DENSE_RANK functions

```
SELECT RANK() OVER (ORDER BY InvoiceTotal) As Rank,  
       DENSE_RANK() OVER (ORDER BY InvoiceTotal)  
             As DenseRank, InvoiceTotal, InvoiceNumber  
FROM Invoices;
```

	Rank	DenseRank	InvoiceTotal	InvoiceNumber
1	1	1	6.00	25022117
2	1	1	6.00	24863706
3	1	1	6.00	24780512
4	4	2	9.95	21-4923721
5	4	2	9.95	21-4748363
6	6	3	10.00	4-321-2596

A query that uses the NTILE function

```
SELECT TermsDescription,  
       NTILE(2) OVER (ORDER BY TermsID) AS Tile2,  
       NTILE(3) OVER (ORDER BY TermsID) AS Tile3,  
       NTILE(4) OVER (ORDER BY TermsID) AS Tile4  
FROM Terms;
```

	TermsDescription	Tile2	Tile3	Tile4
1	Net due 10 days	1	1	1
2	Net due 20 days	1	1	1
3	Net due 30 days	1	2	2
4	Net due 60 days	2	2	3
5	Net due 90 days	2	3	4

The syntax of the analytic functions

```
{FIRST_VALUE|LAST_VALUE} (scalar_expression)
    OVER ([partition_by_clause] order_by_clause
        [rows_range_clause])

{LEAD|LAG} (scalar_expression [, offset [, default]])
    OVER ([partition_by_clause] order_by_clause)

{PERCENT_RANK()|CUME_DIST}
    OVER ([partition_by_clause] order_by_clause)

{PERCENTILE_CONT|PERCENTILE_DISC} (numeric_literal)
    WITHIN GROUP (ORDER BY expression [ASC|DESC])
    OVER (partition_by_clause)
```

The columns in the SalesReps table

Column name	Data type
RepID	int
RepFirstName	varchar(50)
RepLastName	varchar(50)

The columns in the SalesTotals table

Column name	Data type
RepID	int
SalesYear	char(4)
SalesTotal	money

A query that uses the FIRST_VALUE and LAST_VALUE functions

```
SELECT SalesYear, RepFirstName + ' ' +  
       RepLastName AS RepName, SalesTotal,  
       FIRST_VALUE(RepFirstName + ' ' + RepLastName)  
           OVER (PARTITION BY SalesYear  
                  ORDER BY SalesTotal DESC)  
           AS HighestSales,  
       LAST_VALUE(RepFirstName + ' ' + RepLastName)  
           OVER (PARTITION BY SalesYear  
                  ORDER BY SalesTotal DESC  
                  RANGE BETWEEN UNBOUNDED PRECEDING AND  
                  UNBOUNDED FOLLOWING)  
           AS LowestSales  
  FROM SalesTotals JOIN SalesReps  
    ON SalesTotals.RepID = SalesReps.RepID;
```

The result set

	SalesYear	RepName	SalesTotal	HighestSales	LowestSales
1	2009	Jonathon Thomas	1274856.38	Jonathon Thomas	Sonja Martinez
2	2009	Andrew Markasian	1032875.48	Jonathon Thomas	Sonja Martinez
3	2009	Sonja Martinez	978465.99	Jonathon Thomas	Sonja Martinez
4	2010	Andrew Markasian	1132744.56	Andrew Markasian	Lydia Kramer
5	2010	Sonja Martinez	974853.81	Andrew Markasian	Lydia Kramer
6	2010	Jonathon Thomas	923746.85	Andrew Markasian	Lydia Kramer
7	2010	Phillip Winters	655786.92	Andrew Markasian	Lydia Kramer
8	2010	Lydia Kramer	422847.86	Andrew Markasian	Lydia Kramer
9	2011	Jonathon Thomas	998337.46	Jonathon Thomas	Lydia Kramer
10	2011	Sonja Martinez	887695.75	Jonathon Thomas	Lydia Kramer
11	2011	Phillip Winters	72443.37	Jonathon Thomas	Lydia Kramer
12	2011	Lydia Kramer	45182.44	Jonathon Thomas	Lydia Kramer

A query that uses the LAG function

```
SELECT RepID, SalesYear, SalesTotal AS CurrentSales,  
       LAG(SalesTotal, 1, 0)  
          OVER (PARTITION BY RepID ORDER BY SalesYear)  
          AS LastSales,  
       SalesTotal - LAG(SalesTotal, 1, 0)  
          OVER (PARTITION BY REPID ORDER BY SalesYear)  
          AS Change  
FROM SalesTotals;
```

RepID	SalesYear	CurrentSales	LastSales	Change
1	2009	1274856.38	0.00	1274856.38
2	2010	923746.85	1274856.38	-351109.53
3	2011	998337.46	923746.85	74590.61
4	2	978465.99	0.00	978465.99
5	2	974853.81	978465.99	-3612.18
6	2	887695.75	974853.81	-87158.06

A query that uses four more functions

```
SELECT SalesYear, RepID, SalesTotal,  
       PERCENT_RANK() OVER (PARTITION BY SalesYear  
                           ORDER BY SalesTotal) AS PctRank,  
       CUME_DIST() OVER (PARTITION BY SalesYear  
                           ORDER BY SalesTotal) AS CumeDist,  
       PERCENTILE_CONT(.5) WITHIN GROUP (ORDER BY SalesTotal)  
                           OVER (PARTITION BY SalesYear) AS PercentileCont,  
       PERCENTILE_DISC(.5) WITHIN GROUP (ORDER BY SalesTotal)  
                           OVER (PARTITION BY SalesYear) AS PercentileDisc  
FROM SalesTotals;
```

SalesYear	RepID	SalesTotal	PctRank	CumeDist	PercentileCont	PercentileDisc
1	2009	978465.99	0	0.333333333333333	1032875.48	1032875.48
2	2009	1032875.48	0.5	0.6666666666666667	1032875.48	1032875.48
3	2009	1274856.38	1	1	1032875.48	1032875.48
4	2010	422847.86	0	0.2	923746.85	923746.85
5	2010	655786.92	0.25	0.4	923746.85	923746.85
6	2010	923746.85	0.5	0.6	923746.85	923746.85
7	2010	974853.81	0.75	0.8	923746.85	923746.85
8	2010	1132744.56	1	1	923746.85	923746.85
9	2011	45182.44	0	0.25	480069.56	72443.37
10	2011	72443.37	0.333...	0.5	480069.56	72443.37
11	2011	887695.75	0.666...	0.75	480069.56	72443.37
12	2011	998337.46	1	1	480069.56	72443.37

Terms

- Logical functions
- Ranking functions
- Analytic functions