

Database I

SQL (Microsoft T-SQL)

Etelka Szendrői Dr. (PhD)
System and Software Technology Department

szendroi@mik.pte.hu

Categories of T-SQL statements

Data Manipulation Language (DML*)

- Statements for querying and modifying data
- SELECT, INSERT, UPDATE, DELETE

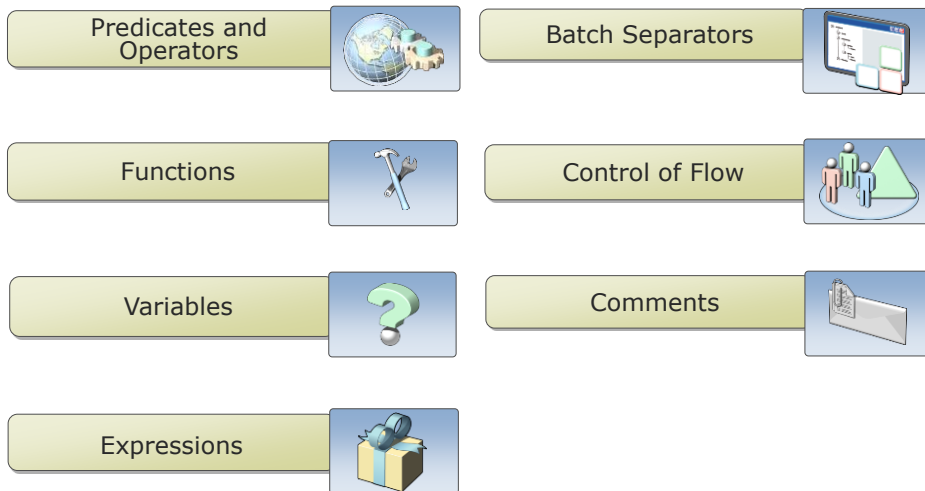
Data Definition Language (DDL)

- Statements for object definitions, create an object, drop an object and modify objects
- CREATE, ALTER, DROP

Data Control Language (DCL)

- Statements for security permissions
- GRANT, REVOKE, DENY

T-SQL language elements



T-SQL language elements: predicates and operators

Elements:	Predicates and Operators:
Predicates	IN, BETWEEN, LIKE
Comparison Operators	=, >, <, >=, <=, <>, !=, !>, !<
Logical Operators	AND, OR, NOT
Arithmetic Operators	+, -, *, /, %
Concatenation	+

T-SQL enforces operator precedence

T-SQL language elements: functions

String Functions	Date and Time Functions	Aggregate Functions
<ul style="list-style-type: none"> • SUBSTRING • LEFT, RIGHT • LEN • DATALENGTH • REPLACE • REPLICATE • UPPER, LOWER • RTRIM, LTRIM 	<ul style="list-style-type: none"> • GETDATE • SYSDATETIME • GETUTCDATE • DATEADD • DATEDIFF • YEAR • MONTH • DAY 	<ul style="list-style-type: none"> • SUM • MIN • MAX • AVG • COUNT

T-SQL language elements: variables

Local variables in T-SQL temporarily store a value of a specific data type

Name begins with single @ sign

@@ reserved for system functions

Assigned a data type

Must be declared and used within the same batch

In SQL Server 2008 and later, can declare and initialize in the same statement

```
DECLARE @MyVar int = 30;
```

T-SQL language elements: expressions

Combination of identifiers, values, and operators evaluated to obtain a single result

Can be used in SELECT statements

- SELECT clause

- WHERE clause

Can be single constant, single-valued function, or variable

Can be combined if expressions have same the data type

```
SELECT YEAR(OrderDate) + 1 ...
```

```
SELECT OrderQty * UnitPrice ...
```

T-SQL language elements: batch separators

Batches are sets of commands sent to SQL Server as a unit

Batches determine variable scope, name resolution

To separate statements into batches, use a separator:

- SQL Server tools use the GO keyword

- GO is not a SQL Server T-SQL command

T-SQL language elements: control of flow, errors, and transactions

Allow you to control the flow of execution within code, handle errors, and maintain transactions

Used in programmatic code objects

Stored procedures, triggers, statement blocks

Control of Flow	Error Handling	Transaction Control
<ul style="list-style-type: none"> • IF...ELSE • WHILE • BREAK • CONTINUE • BEGIN...END 	<ul style="list-style-type: none"> • TRY...CATCH 	<ul style="list-style-type: none"> • BEGIN TRANSACTION • COMMIT TRANSACTION • ROLLBACK TRANSACTION

T-SQL language elements: comments

Marks T-SQL code as a comment:

For a block, enclose it between `/*` and `*/` characters

```
/*
    This is a block
    of commented code
*/
```

T-SQL Editors such as SSMS will typically color-code comments, as shown above

```
-- This line of text will be ignored
```

Elements of the SELECT statement

Clause	Expression
SELECT	<select list>
FROM	<table source>
WHERE	<search condition>
GROUP BY	<group by list>
ORDER BY	<order by list>

Logical query processing

The order in which a query is written is not the order in which it is evaluated by SQL Server.

5: SELECT <select list>

1: FROM <table source>

2: WHERE <search condition>

3: GROUP BY <group by list>

4: HAVING <search condition>

6: ORDER BY <order by list>



Basic SELECT Statements

Retrieving columns from a table or view

Use SELECT with column list to display columns

Use FROM to specify a source table or view

Specify both schema and table names

Delimit names if necessary

End all statements with a semicolon

Keyword	Expression
SELECT	<select list>
FROM	<table source>

```
SELECT CustomerID, StoreID  
FROM Sales.Customer;
```

Using calculations in the SELECT clause

Calculations are scalar, returning one value per row

Operator	Description
+	Add or concatenate
-	Subtract
*	Multiply
/	Divide
%	Modulo

Using scalar expressions in the SELECT clause

```
SELECT unitprice, OrderQty, (unitprice * OrderQty)
FROM sales.salesorderdetail;
```

Understanding DISTINCT

Specifies that only unique rows can appear in the result set

Removes duplicates based on column list results, not source table

Provides uniqueness across set of selected columns

Removes rows already operated on by WHERE, HAVING, and GROUP BY clauses

Some queries may improve performance by filtering out duplicates prior to execution of SELECT clause

SELECT DISTINCT syntax

```
SELECT DISTINCT <column list>
FROM <table or view>
```

```
SELECT DISTINCT StoreID
FROM Sales.Customer;
```

```
StoreID
-----
1234
570
902
1898
710
```

Using aliases to refer to columns

Column aliases using AS

```
SELECT SalesOrderID, UnitPrice, OrderQty AS Quantity
FROM Sales.SalesOrderDetail;
```

Column aliases using =

```
SELECT SalesOrderID, UnitPrice, Quantity = OrderQty
FROM Sales.SalesOrderDetail;
```

Accidental column aliases

```
SELECT SalesOrderID, UnitPrice Quantity
FROM Sales.SalesOrderDetail;
```

Using aliases to refer to tables

Create table aliases in the FROM clause using AS

```
SELECT SalesOrderID, ProductID  
FROM Sales.SalesOrderDetail AS SalesOrders;
```

Table aliases without AS

```
SELECT SalesOrderID, ProductID  
FROM Sales.SalesOrderDetail SalesOrders;
```

Using table aliases in the SELECT clause

```
SELECT SalesOrders.SalesOrderID, SalesOrders.ProductID  
FROM Sales.SalesOrderDetail AS SalesOrders;
```

Filtering and Sorting Data

Using the ORDER BY clause

ORDER BY sorts rows in results for presentation purposes

- Use of ORDER BY guarantees the sort order of the result

- Last clause to be logically processed

- Sorts all NULLs together

ORDER BY can refer to:

- Columns by name, alias or ordinal position (not recommended)

- Columns not part of SELECT list unless DISTINCT clause specified

Declare sort order with ASC or DESC

ORDER BY clause examples

ORDER BY with column names:

```
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
ORDER BY OrderDate;
```

ORDER BY with column alias:

```
SELECT SalesOrderID, CustomerID,
YEAR(OrderDate) AS OrderYear
FROM Sales.SalesOrderHeader
ORDER BY OrderYear;
```

ORDER BY with descending order:

```
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
ORDER BY OrderDate DESC;
```

Filtering data in the WHERE clause

WHERE clauses use predicates

- Must be expressed as logical conditions

- Only rows for which predicate evaluates to TRUE are accepted

- Values of FALSE or UNKNOWN are filtered out

WHERE clause follows FROM, precedes other clauses

- Can't see aliases declared in SELECT clause

Can be optimized by SQL Server to use indexes

WHERE clause syntax

Filter rows for customers in territory 6

```
SELECT CustomerID, TerritoryID
FROM Sales.Customer
WHERE TerritoryID = 6;
```

Filter rows for orders in territories greater than or equal to 6

```
SELECT CustomerID, TerritoryID
FROM Sales.Customer
WHERE TerritoryID >= 6;
```

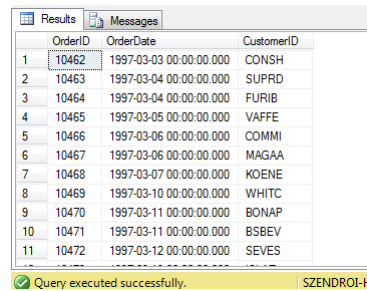
Filter orders within a range of dates

```
SELECT CustomerID, TerritoryID, StoreID
FROM Sales.Customer
WHERE StoreID >= 1000 AND StoreID <= 1200;
```

Example

List the orders of month 3 in 1997

```
SELECT OrderID, OrderDate, CustomerID
FROM Orders
WHERE YEAR(OrderDate)=1997 AND MONTH(OrderDate)=3;
```



	OrderID	OrderDate	CustomerID
1	10462	1997-03-03 00:00:00.000	CONSH
2	10463	1997-03-04 00:00:00.000	SUPRD
3	10464	1997-03-04 00:00:00.000	FURIB
4	10465	1997-03-05 00:00:00.000	VAFFE
5	10466	1997-03-06 00:00:00.000	COMMI
6	10467	1997-03-06 00:00:00.000	MAGAA
7	10468	1997-03-07 00:00:00.000	KOENE
8	10469	1997-03-10 00:00:00.000	WHITC
9	10470	1997-03-11 00:00:00.000	BONAP
10	10471	1997-03-11 00:00:00.000	BSBEV
11	10472	1997-03-12 00:00:00.000	SEVES

Query executed successfully. SZENDROI-H

Filtering data in the SELECT clause

TOP allows you to limit the number or percentage of rows returned. Works with ORDER BY clause to limit rows by sort order.

If ORDER BY list is not unique, results are not deterministic (no single correct result set).

Modify ORDER BY list to ensure uniqueness, or use TOP WITH TIES.

Added to SELECT clause:

SELECT TOP (N) | TOP (N) Percent

With percent, number of rows rounded up.

SELECT TOP (N) WITH TIES

Retrieve duplicates where applicable (nondeterministic).

TOP is proprietary to Microsoft SQL Server.

Filtering using TOP

Filter rows for customers to display top 20 TotalDue items

```
SELECT TOP (20) SalesOrderID, CustomerID,
TotalDue
FROM Sales.SalesOrderHeader
ORDER BY TotalDue DESC;
```

Filter rows for customers to display top 20 TotalDue items with ties

```
SELECT TOP (20) WITH TIES SalesOrderID,
CustomerID, TotalDue
FROM Sales.SalesOrderHeader
ORDER BY TotalDue DESC;
```

Filter rows for customers to display top 1% of TotalDue items

```
SELECT TOP (1) PERCENT SalesOrderID, CustomerID,
TotalDue
FROM Sales.SalesOrderHeader
ORDER BY TotalDue DESC;
```

List the code and unitPrice value of the five most expensive products belonging to the category 1.

```
SELECT TOP (5) ProductID, UnitPrice
FROM Products
WHERE CategoryID = 1
ORDER BY UnitPrice DESC;
```

	ProductID	UnitPrice
1	38	263.50
2	43	46.00
3	2	19.00
4	1	18.00
5	35	18.00

```
SELECT TOP (5) WITH TIES ProductID, UnitPrice
FROM Products
WHERE CategoryID = 1
ORDER BY UnitPrice DESC;
```

	ProductID	UnitPrice
1	38	263.50
2	43	46.00
3	2	19.00
4	1	18.00
5	39	18.00
6	35	18.00
7	76	18.00

```
SELECT TOP (5) ProductID, UnitPrice
FROM Products
WHERE CategoryID = 1
ORDER BY UnitPrice DESC, ProductID DESC;
```

	ProductID	UnitPrice
1	38	263.50
2	43	46.00
3	2	19.00
4	76	18.00
5	39	18.00

Handling NULL in queries

Different components of SQL Server handle NULL differently
 Query filters (ON, WHERE, HAVING) filter out UNKNOWNs
 CHECK constraints accept UNKNOWNs
 ORDER BY, DISTINCT treat NULLs as equals

Testing for NULL

Use IS NULL or IS NOT NULL rather than = NULL or <> NULL

```
SELECT CustomerID, StoreID, TerritoryID
FROM Sales.Customer
WHERE StoreID IS NULL
ORDER BY TerritoryID
```

Common built-in aggregate functions

Common	Statistical	Other
<ul style="list-style-type: none"> • SUM • MIN • MAX • AVG • COUNT • COUNT_BIG 	<ul style="list-style-type: none"> • STDEV • STDEVP • VAR • VARP 	<ul style="list-style-type: none"> • CHECKSUM_AGG • GROUPING • GROUPING_ID

Working with aggregate functions

Aggregate functions:

- Return a scalar value (with no column name)

- Ignore NULLs except in COUNT(*)

- Can be used in

- SELECT, HAVING, and ORDER BY clauses

- Frequently used with GROUP BY clause

```
SELECT COUNT (DISTINCT SalesOrderID) AS
UniqueOrders,
AVG(UnitPrice) AS Avg_UnitPrice,
MIN(OrderQty) AS Min_OrderQty,
MAX(LineTotal) AS Max_LineTotal
FROM Sales.SalesOrderDetail;
```

UniqueOrders	Avg_UnitPrice	Min_OrderQty	Max_LineTotal
31465	465.0934	1	27893.619000

Using DISTINCT with aggregate functions

Use DISTINCT with aggregate functions to summarize only unique values

DISTINCT aggregates eliminate duplicate values, not rows (unlike SELECT DISTINCT)

Compare (with partial results):

```
SELECT SalesPersonID, YEAR(OrderDate) AS OrderYear,
COUNT(CustomerID) AS All_Custs,
COUNT(DISTINCT CustomerID) AS Unique_Custs
FROM Sales.SalesOrderHeader
GROUP BY SalesPersonID, YEAR(OrderDate);
```

SalesPersonID	OrderYear	All_Custs	Unique_custs
289	2006	84	48
281	2008	52	27
285	2007	9	8
277	2006	140	57

Using the GROUP BY clause

GROUP BY creates groups for output rows, according to unique combination of values specified in the GROUP BY clause

```
SELECT <select_list>  
FROM <table_source>  
WHERE <search_condition>  
GROUP BY <group_by_list>;
```

GROUP BY calculates a summary value for aggregate functions in subsequent phases

```
SELECT SalesPersonID, COUNT(*) AS Cnt  
FROM Sales.SalesOrderHeader  
GROUP BY SalesPersonID;
```

Detail rows are "lost" after GROUP BY clause is processed

GROUP BY and HAVING

GROUP BY and logical order of operations

HAVING, SELECT, and ORDER BY must return a single value per group

All columns in SELECT, HAVING, and ORDER BY must appear in GROUP BY clause or be inputs to aggregate expressions

Logical Order	Phase	Comments
5	SELECT	
1	FROM	
2	WHERE	
3	GROUP BY	Creates groups
4	HAVING	Operates on groups
6	ORDER BY	

Using GROUP BY with aggregate functions

Aggregate functions are commonly used in SELECT clause, summarize per group:

```
SELECT CustomerID, COUNT(*) AS cnt
FROM Sales.SalesOrderHeader
GROUP BY CustomerID;
```

Aggregate functions may refer to any columns, not just those in GROUP BY clause

```
SELECT productid, MAX(OrderQty) AS largest_order
FROM Sales.SalesOrderDetail
GROUP BY productid;
```

Filtering grouped data using HAVING Clause

HAVING clause provides a search condition that each group must satisfy

HAVING clause is processed after GROUP BY

```
SELECT CustomerID, COUNT(*) AS
Count_Orders
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
HAVING COUNT(*) > 10;
```

Compare HAVING to WHERE clauses

- Using a COUNT(*) expression in HAVING clause is useful to solve common business problems:
- Show only customers that have placed more than one order:

```
SELECT Cust.Customerid, COUNT(*) AS cnt
FROM Sales.Customer AS Cust
JOIN Sales.SalesOrderHeader AS Ord ON Cust.CustomerID =
ORD.CustomerID
GROUP BY Cust.CustomerID
HAVING COUNT(*) > 1;
```

- Show only products that appear on 10 or more orders:

```
SELECT Prod.ProductID, COUNT(*) AS cnt
FROM Production.Product AS Prod
JOIN Sales.SalesOrderDetail AS Ord ON Prod.ProductID =
Ord.ProductID
GROUP BY Prod.ProductID
HAVING COUNT(*) >= 10;
```



Data Types

SQL Server data types

SQL Server associates columns, expressions, variables, and parameters with data types

Data types determine what kind of data can be stored in the field:

Integers, characters, dates, money, binary strings, etc.

SQL Server supplies several built-in data types

Developers can also define custom types

Aliases in T-SQL

User-defined types in .NET code

Built-in data types are categorized as shown in the table below

SQL Server Data Type Categories	
Exact numeric	Unicode characters
Approximate numeric	Binary strings
Date and time	Other
Character strings	

Character data types

SQL Server supports two kinds of character data types:

Regular: CHAR, VARCHAR

One byte stored per character

Only 256 possible characters – limits language support

Unicode: NCHAR, NVARCHAR

Two bytes stored per character

65k characters represented – multiple language support

Precede characters with N' (National)

TEXT, NTEXT deprecated

Use VARCHAR(MAX), NVARCHAR(MAX) instead

Character data types

- CHAR, NCHAR are fixed length
- VARCHAR, NVARCHAR are variable length
- Character data is delimited with single quotes

Data Type	Range	Storage
CHAR(n), NCHAR(n)	1-8000 characters	n bytes, padded 2*n bytes, padded
VARCHAR(n), NVARCHAR(n)	1-8000 characters	n+2 bytes (2*n) +2 bytes
VARCHAR(MAX), NVARCHAR(MAX)	1-2 ³¹ -1 characters	Actual length + 2

Character Data Types

SQL STANDARD	ORACLE 11G	DB2 9.7	MS SQL SERVER 2008
CHARACTER	CHARACTER	CHARACTER	CHARACTER
VARYINGVARCHAR	VARYING VARCHAR VARCHAR2 LONG VARCHAR	VARYING VARCHAR LONG VARCHAR	VARYING VARCHAR TEXT
CLOB or CHARACTER LARGE OBJECT	CLOB	CLOB	VARCHAR (MAX)
NCHAR	NCHAR	GRAPHIC	NCHAR
NCHAR VARYING (n)	NCHAR VARYING NVARCHAR2	VARGRAPHIC LONG VARGRAPHIC	NVARCHAR
NATIONALCHARACTER LARGE OBJECT	NCLOB	DBCLOB	NVARCHAR (MAX)

43

Character Data Types

SQL STANDARD	POSTGRESQL	MYSQL	MS ACCESS	HSQldb (OPENOFFICE BASE)
CHARACTER	CHARACTER	CHAR	TEXT	CHARACTER
VARYINGVARCHAR	VARCHAR		TEXT	VARCHAR
CLOB or CHARACTER LARGE OBJECT	TEXT	LONGTEXT MEDIUMTEXT TINYTEXT	MEMO	LONGVARCHAR OBJECT
NCHAR	VARCHAR TEXT	VARCHAR LONGTEXT MEDIUMTEXT TINYTEXT	TEXT MEMO	CHARACTER
NCHAR VARYING	VARCHAR	VARCHAR LONGTEXT MEDIUMTEXT TINYTEXT	TEXT MEMO	VARCHAR LONGVARCHAR
NATIONALCHARACTER LARGE OBJECT	VARCHAR TEXT	VARCHAR LONGTEXT	TEXT MEMO	LONGVARCHAR OBJECT

44

Numeric data types

- Decimal/numeric are functionally equivalent and use precision and scale parameters:

```
DECLARE @mydecimal AS DECIMAL(8,2)
```

- Approximate Numeric

Data Type	Range	Storage (bytes)
float(n)	- 1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308	Depends on value of n, 4 or 8 bytes
real	- 3.40E + 38 to -1.18E - 38, 0 and 1.18E - 38 to 3.40E + 38	4

- float(24) is the ISO synonym for float
 - In float(n), n is the number of bytes used to store the mantissa of the float number in scientific notation
- Values of float are truncated when converted to integer types

Numeric data types

Exact Numeric

Data type	Range	Storage (bytes)
tinyint	0 to 255	1
smallint	-32,768 to 32,768	2
int	2 ³¹ (-2,147,483,648) to 2 ³¹ -1 (2,147,483,647)	4
Bigint	-2 ⁶³ - 2 ⁶³ -1 (+/- 9 quintillion)	8
bit	1, 0 or NULL	1
decimal/numeric	- 10 ³⁸ + 1 through 10 ³⁸ - 1 when maximum precision is used	5-17
money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8
smallmoney	- 214,748.3648 to 214,748.3647	4

Numeric Data Types (Exact numeric)

SQL STANDARD	ORACLE 11G	DB2 9.5	MS SQL SERVER 2008
INTEGER	NUMBER (38) INT	INTEGER BIGINT	INTEGER BIGINT
SMALLINT	SMALLINT NUMBER (38)	SMALLINT	SMALLINT TINYINT
NUMERIC	NUMERIC DECIMAL NUMBER	NUMERIC DECIMAL	NUMERIC DECIMAL MONEY SMALLMONEY

SQL STANDARD	POSTGRESQL	MYSQL	MS ACCESS	HSQldb (OPENOFFICE BASE)
INTEGER	INTEGER BIGINT	INTEGER BIGINT	NUMBER (INTEGER, LONG INTEGER)	INTEGER BIGINT
SMALLINT	SMALLINT	SMALLINT TINYINT	NUMBER (INTEGER)	SMALLINT TINYINT
NUMERIC	NUMERIC	NUMERIC	NUMBER	NUMERIC

47

Numeric Data Types (approximate numeric)

SQL STANDARD	ORACLE 11G	DB2 9.5	MS SQL SERVER 2008
FLOAT	FLOAT NUMBER	FLOAT	FLOAT
REAL	REAL NUMBER	REAL	REAL
DOUBLE PRECISION	DOUBLE PRECISION NUMBER	DOUBLE PRECISION	DOUBLE PRECISION

SQL STANDARD	POSTGRESQL	MYSQL	MS ACCESS	HSQldb (OPENOFFICE BASE)
FLOAT	FLOAT	FLOAT	NUMBER (DECIMAL)	FLOAT
REAL	REAL	REAL	NUMBER (DECIMAL)	REAL
DOUBLE PRECISION	DOUBLE	DOUBLE	NUMBER (DOUBLE)	DOUBLE

48

Data Types and Range

DATA TYPE	STORAGE SIZE (BYTES)	RANGE	NOTES
INTEGER	4	-2,147,483,648 to +2,147,483,647	Implemented in all RDBMSs
TINYINT	1	0 through 255	MS SQL Server only
SMALLINT	2	-32,768 to +32,768	Implemented in all RDBMSs
BIGINT	8	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,808	Implemented in all RDBMSs
MONEY	8	- 922,337,203,685,477.5808 to + 922,337,203,685,477.5807	MS SQL Server only
SMALLMONEY	4	- 214,748.3648 to + 214,748.3647	MS SQL Server only
REAL	4	The range is from negative 3.402E + 38 to negative 1.175E - 37, or from positive 1.175E - 37 to 3.402E + 38. It also includes 0.	Implemented in all RDBMSs
FLOAT	4 to 8	The number can be zero or can range from -1.79769E + 308 to -2.225E - 307, or from 2.225E - 307 to 1.79769E + 308.	Implemented in all RDBMSs (if only as synonyms)
DOUBLE	8	The number can be zero or can range from -1.79769E + 308 to -2.225E - 307, or from 2.225E - 307 to 1.79769E + 308.	Implemented in all RDBMSs

49

Character Data Types

String concatenation

SQL Server uses the + (plus) sign to concatenate characters:

Concatenating a value with a NULL returns a NULL

```
SELECT BusinessEntityID, FirstName, LastName,
       FirstName + N' ' + LastName AS FullName
FROM Person.Person;
```

SQL Server 2012 introduces CONCAT() function

Converts NULL to empty string before concatenation

```
SELECT AddressLine1, City, StateProvinceID,
       CONCAT(AddressLine1, ', ' + City, ', ' + PostalCode) AS
Location
FROM Person.Address
```

Character string functions

Common functions that modify character strings

Function	Syntax	Remarks
SUBSTRING()	SUBSTRING (expression , start , length)	Returns part of an expression
LEFT(), RIGHT()	LEFT (expression , integer_value) RIGHT (expression , integer_value)	LEFT() returns left part of string up to integer_value. RIGHT() returns right part of string.
LEN(), DATALENGTH()	LEN (string_expression) DATALENGTH (expression)	LEN() returns the number of characters of the specified string expression, excluding trailing blanks. DATALENGTH() returns the number bytes used.
CHARINDEX()	CHARINDEX (expressionToFind, expressionToSearch)	Searches an expression for another expression and returns its starting position if found. Optional start position.
REPLACE()	REPLACE (string_expression , string_pattern , string_replacement)	Replaces all occurrences of a specified string value with another string value.
UPPER(), LOWER()	UPPER (character_expression) LOWER (character_expression)	UPPER() returns a character expression with lowercase character data converted to uppercase. LOWER() converts uppercase to lowercase.

The LIKE predicate

The LIKE predicate used to check a character string against a pattern

Patterns expressed with symbols

% (Percent) represents a string of any length

_ (Underscore) represents a single character

[<List of characters>] represents a single character within the supplied list

[<Character> - <character>] represents a single character within the specified range

[^<Character list or range>] represents a single character not in the specified list or range

ESCAPE Character allows you to search for a character that is also a wildcard character (% , _ [] for example)

```
SELECT ProductLine, Name, ProductNumber  
FROM Production.Product  
WHERE Name LIKE 'Mountain%'
```

Date and Time Data Types

Date and time data types

Older versions of SQL Server supported only DATETIME and SMALLDATETIME

DATE, TIME, DATETIME2, and DATETIMEOFFSET introduced in SQL Server 2008

Data Type	Storage (bytes)	Date Range	Accuracy	Recommended Entry Format
DATETIME	8	January 1, 1753 to December 31, 9999	3-1/3 milliseconds	'YYMMDD hh:mm:ss:nnn'
SMALLDATETIME	4	January 1, 1900 to June 6, 2079	1 minute	'YYMMDD hh:mm:ss:nnn'
DATETIME2	6 to 8	January 1, 0001 to December 31, 9999	100 nanoseconds	'YYMMDD hh:mm:ss:nnnnnn'
DATE	3	January 1, 0001 to December 31, 9999	1 day	'YYYY-MM-DD'
TIME	3 to 5		100 nanoseconds	'hh:mm:ss:nnnnnnn'
DATETIMEOFFSET	8 to 10	January 1, 0001 to December 31, 9999	100 nanoseconds	'YY-MM-DD hh:mm:ss:nnnnnnn [+ -]hh:mm'

Date and time data types: literals

Data Type	Language-Neutral Formats	Examples
DATETIME	'YYMMDD hh:mm:ss:nnn' 'YYYY-MM-DDThh:mm:ss:nnn' 'YYMMDD'	'20120212 12:30:15.123' '2012-02-12T12:30:15.123' '20120212'
SMALLDATETIME	'YYMMDD hh:mm' 'YYYY-MM-DDThh:mm' 'YYMMDD'	'20120212 12:30' '2012-02-12T12:30' '20120212'
DATETIME2	'YYYY-MM-DD' 'YYMMDD hh:mm:ss:nnnnnnn' 'YYYY-MM-DD hh:mm:ss:nnnnnnn' 'YYYY-MM-DDThh:mm:ss:nnnnnnn' 'YYMMDD' 'YYYY-MM-DD'	'20120212 12:30:15.1234567' '2012-02-12 12:30:15.1234567' '2012-02-12T12:30:15.1234567' '20120212' '2012-02-12'
DATE	'YYMMDD' 'YYYY-MM-DD'	'20120212' '2012-02-12'
TIME	'hh:mm:ss:nnnnnnn'	'12:30:15.1234567'
DATETIMEOFFSET	'YYMMDD hh:mm:ss:nnnnnnn [+ -]hh:mm' 'YYYY-MM-DD hh:mm:ss:nnnnnnn [+ -]hh:mm' 'YYMMDD' 'YYYY-MM-DD'	'20120212 12:30:15.1234567 +02:00' '2012-02-12 12:30:15.1234567 +02:00' '20120212' '2012-02-12'



Working with date and time separately

DATETIME, SMALLDATETIME, DATETIME2, and DATETIMEOFFSET include both date and time data

If only date is specified, time set to midnight (all zeroes)

If only time is specified, date set to base date (January 1, 1900)

```
DECLARE @DateOnly DATETIME = '20120212';
SELECT @DateOnly;
```

```
RESULT
-----
2012-02-12 00:00:00.000
```

Querying date and time values

Date values converted from character literals often omit time

Queries written with equality operator for date will match midnight

```
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate = '20070825';
```

If time values are stored, queries need to account for time past midnight on a date

Use range filters instead of equality

```
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate >= '20070825'
AND OrderDate < '20070826';
```

Date and time functions

Function	Syntax	Remarks
DATEADD()	DATEADD(datepart, interval, date)	Adds interval to date, returns same datatype as date
EOMONTH()	EOMONTH(start_date, interval)	Returns last day of month as start date, with optional offset
SWITCHOFFSET()	SWITCHOFFSET(datetimeoffset, time_zone)	Changes time zone offset
TODATETIMEOFFSET()	TODATETIMEOFFSET(expression, time_zone)	Converts datetime2 into datetimeoffset

```
SELECT DATEADD(day,1, '20120212');
SELECT EOMONTH('20120212');
```

Converting with CAST

Converts a value from one data type to another

Can be used in SELECT and WHERE clauses

ANSI standard

Truncation can occur if converting to smaller data type

CAST Example:

```
SELECT CAST(SYSDATETIME() AS date) AS 'TodaysDate';
```

Returns an error if data types are incompatible:

```
--attempt to convert datetime2 to int
SELECT CAST(SYSDATETIME() AS int);
```

```
Msg 529, Level 16, State 2, Line 1
Explicit conversion from data type datetime2 to int is
not allowed.
```

Converting with CONVERT

Converts a value from one data type to another

Can be used in SELECT and WHERE clauses

CONVERT is specific to SQL Server, not standards-based

Style specifies how input value is converted:

Date, time, numeric, XML, etc.

Example:

```
SELECT CONVERT(CHAR(8), CURRENT_TIMESTAMP,112) AS ISO_style;
```

```
ISO_style  
-----  
20120212
```

Thank you for your attention!