Collation decides language support, case sensitivity & sort order for character data

https://mva.microsoft.com/en-US/training-courses/querying-microsoft-sql-server-2012-databases-jump-start-8241

--ddl or data definition language: create alter drop
--dml or data manipulation language: select insert update delete (CRUD operations)
--dcl or data control language: grant revoke deny

multiple data files might result in increased performance.

T-SQL enforces operator precedence

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

? exists → standards

but use concat as it handles null

I think it gives us number of bytes

T-SQL functions:

String functions: SUBSTRING,LEFT, RIGHT,LEN,DATALENGTH, REPLACE,REPLICATE,UPPER, LOWER,RTRIM, LTRIM
Date and time functions: GETDATE,SYSTDATETIME,GETUTCDATE,DATEADD,DATEDIFF,YEAR,MONTH,DAY
Aggregate functions: SUM,MIN,MAX,AVG,COUNT

→ returns datetime2. should I use this (or) current_timestamp ?? What is the standard?

T-SQL 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;
```

AS

T-SQL 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 ...
```

Predicates → logical expression (or) logical condition.

T-SQL 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 flow control, errors and transactions:

Used in programmatic code objects such as stored procedures, triggers, statement blocks
Flow control: IF...ELSE,WHILE,BREAK,CONTINUE,BEGIN...END
Error handling: TRY...CATCH
Transaction control: BEGIN TRANSACTION,COMMIT TRANSACTION,ROLLBACK TRANSACTION

HAVING clause can used aggregate funcs as the input has been GROUPED

The order in which a query is written is not the order in which it is evaluated by the server. The order is:

5: SELECT <select list>
1: FROM <table source>
2: WHERE <search condition>
3: GROUP BY <group by list>
4: HAVING <search condition>

predicates (logical expressions) & their relation with propositions

logical query processing phases are:-

predicates

6: ORDER BY <order by list>

```
USE AdventureWorks2014;

SELECT SalesPersonID, YEAR(OrderDate) AS OrderYear
FROM Sales.SalesOrderHeader
WHERE CustomerID = 29974
GROUP BY SalesPersonID, YEAR(OrderDate)
HAVING COUNT(*) > 1
ORDER BY SalesPersonID, OrderYear;
```

1 FROM
2 WHERE
3 GROUP BY
4 HAVING
5 SELECT
   1. EXPRESSIONS
   2. DISTINCT
6 ORDER BY
   1. TOP / OFFSET-FETCH

When performance tuning, using `SELECT 1` gives you stats for just speaking to sql server on network(rather than what happens inside the database engine)

Advanced SELECT clauses (DISTINCT, aliases, CASE, and scalar functions)

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

either the 'DISTINCT' clause is used before start of column list (or) used inside aggregating funcs anywhere if used with group by

In certain cases GROUP BY can be use to the same affect as DISTINCT

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

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

Using aliases to refer to columns: only 'AS' is the standard. Rest are here for legacy reasons

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

Using aliases to refer to tables: using alias makes name shorter and also then u can use same table multiple times(with different aliases) in joins

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

T-SQL case expressions:

Simple CASE
- Compares one value to a list of possible values and returns first match
- If no match, returns value found in optional ELSE clause
- If no match and no ELSE, returns NULL

Searched CASE
- Evaluates a set of predicates, or logical expressions
- Returns value found in THEN clause matching first expression that evaluates to TRUE

T-SQL CASE expressions return a single (scalar) value
CASE expressions may be used in:
- SELECT column list (behaves as calculated column requiring an alias)
- WHERE or HAVING clauses
- ORDER BY clause

```
SELECT ProductID, Name, ProductSubCategoryID,
  CASE ProductSubCategoryID
    WHEN 1 THEN 'Beverages'
    ELSE 'Unknown Category'
  END
FROM Production.Product
```

Joins:

Join Type	Description
Cross	Combines all rows in both tables (creates Cartesian product).
Inner(the default join)	Starts with Cartesian product; applies filter to match rows between tables based on predicate.
Outer	Starts with Cartesian product; all rows from designated table preserved, matching rows from other table retrieved. Additional NULLs inserted as placeholders.

Inner Join: As it is the default join, just specify join for inner join
Returns only rows where a match is found in both tables
Matches rows based on attributes supplied in predicate
    ON clause in SQL-92 syntax
Why filter in ON clause?
    Logical separation between filtering for purposes of JOIN and filtering results in WHERE
    'on' filter is applied before the 'where' filter
    Typically no difference to query optimizer
If JOIN predicate operator is =, also known as equi-join

```
SELECT SOH.SalesOrderID, SOH.OrderDate, SOD.ProductID, SOD.UnitPrice, SOD.OrderQty
FROM Sales.SalesOrderHeader AS SOH
JOIN Sales.SalesOrderDetail AS SOD
ON SOH.SalesOrderID = SOD.SalesOrderID;
```

Outer Join:
Returns all rows from one table and any matching rows from second table
One table's rows are "preserved"
    Designated with LEFT, RIGHT, FULL keyword
    All rows from preserved table output to result set
Matches from other table retrieved
Additional rows added to results for non-matched rows
    NULLs added in place where attributes do not match
Example: Return all customers and for those who have placed orders, return order information. Customers without matching orders will display NULL for order details.

Customers that did not place orders

```
SELECT CUST.CustomerID, CUST.StoreID, ORD.SalesOrderID, ORD.OrderDate
FROM Sales.Customer AS CUST
LEFT OUTER JOIN Sales.SalesOrderHeader AS ORD
ON CUST.CustomerID = ORD.CustomerID
WHERE ORD.SalesOrderID IS NULL;
```

Cross join:
Combine each row from first table with each row from second table
All possible combinations are displayed
Logical foundation for inner and outer joins
    INNER JOIN starts with Cartesian product, adds filter
    OUTER JOIN takes Cartesian output, filtered, adds back non-matching rows (with NULL placeholders)
Due to Cartesian product output, not typically a desired form of JOIN
Some useful exceptions:
    Generating a table of numbers for testing

Example:
Create test data by returning all combinations of two inputs:

```
SELECT EMP1.BusinessEntityID, EMP2.JobTitle
FROM HumanResources.Employee AS EMP1
CROSS JOIN HumanResources.Employee AS EMP2;
```

Self Join:

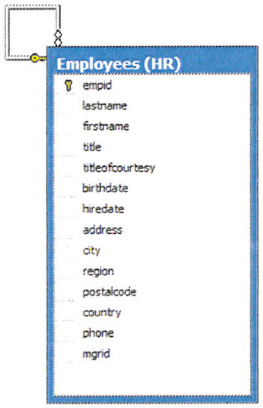
Why use self-joins?

Compare rows in same table to each other

Create two instances of same table in FROM clause

At least one alias required

Example: Return all employees and the name of the employee's manager



Return all employees with ID of employee's manager when a manager exists (INNER JOIN):

```
SELECT  EMP.EmpID, EMP.LastName, EMP.JobTitle, EMP.MgrID, MGR.LastName
FROM    HR.Employees AS EMP
INNER JOIN HR.Employees AS MGR
ON EMP.MgrID = MGR.EmpID;
```

Return all employees with ID of manager (OUTER JOIN). This will return NULL for the CEO:

```
SELECT  EMP.EmpID, EMP.LastName,
        EMP.Title, MGR.MgrID
FROM HumanResources.Employee AS EMP
LEFT OUTER JOIN HumanResources.Employee AS MGR
ON EMP.MgrID = MGR.EmpID;
```

set theory. order does not matter

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(processed after SELECT)

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

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

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 → keep predicate is search argument form.

```
SELECT CustomerID, TerritoryID
FROM Sales.Customer
```

```
WHERE TerritoryID = 6;
SELECT CustomerID, TerritoryID
FROM Sales.Customer
WHERE TerritoryID >= 6;
SELECT CustomerID, TerritoryID, StoreID
FROM Sales.Customer
WHERE StoreID >= 1000 AND StoreID <= 1200;
```

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

TOP is not standard. use offset fetch.

By using unique key or primary key in 'order by' list.

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

--this might show u 23 rows
SELECT TOP (20) WITH TIES SalesOrderID, CustomerID, TotalDue
FROM Sales.SalesOrderHeader
ORDER BY TotalDue DESC;
SELECT TOP (1) PERCENT SalesOrderID, CustomerID, TotalDue
FROM Sales.SalesOrderHeader
ORDER BY TotalDue DESC;
```

Offset: could be used for paging...It offsets the result set by the number of records specified. Here we are getting rows 21 to 30.

```
select * from Production.Product
order by ListPrice, ProductID
offset 20 rows
fetch next 10 rows only;
```

only 'offset-fetch' clause requires 'order by' clause, not 'TOP' clause. But we can workaround that by providing a dummy 'order by Null' clause.

Union: if only say union, then the combined result set would have distinct records. UNION ALL would remove duplicates

```
--i had to use wrapper selects as otherwise it threw an error
select * from (
select top(2) Name,ListPrice,Color  from Production.Product as p
where Color='Black'
order by ListPrice desc
) as a
Union all
select * from(
select top(2) Name,ListPrice,Color from Production.Product as p
where Color='Red'
order by ListPrice desc
) as b
```

why did we had to use a table expression here??

Because 'order by' is not allowed in input queries for set operators. So either use derived table expressions as used here (or) use 'order by' once for the result.

beyond: distinct rows, unique rows, not duplicate rows

Null values:

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 & GROUP BY as well

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
```

UNIQUE constraint will only allow one NULL value in the column.

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	

Approximate Numeric types:

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

Exact Numeric types:

Data type	Range	Storage (bytes)
tinyint	0 to 255	1
smallint	-32,768 to 32,767	2
int	2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)	4
Bigint	-2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)	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

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

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

8 refers to total number of digits

2 " " number of digits after decimal

Note that the bit field still uses 1 byte. Strange.

Binary String data types: for storing blobs. Say image data.

Data Type	Range	Storage (bytes)
binary(n)	1-8000 bytes	n bytes
varbinary(n)	1-8000 bytes	actual length + 2
varbinary(MAX)	1-2.1 billion (approx) bytes	actual length + 2

→ IMAGE data type deprecated use VARBINARY(MAX)

Other data types:

Data Type	Range	Storage (bytes)	Remarks
rowversion	Auto-generated	8	Successor type to timestamp
uniqueidentifier	Auto-generated	16	Globally unique identifier (GUID)
xml	0-2 GB	0-2 GB	Stores XML in native hierarchical structure
cursor	N/A	N/A	Not a storage data type
hierarchyid	N/A	Depends on content	Represents position in a hierarchy
sql_variant	0-8000 bytes	Depends on content	Can store data of various data types
table	N/A	N/A	Not a storage data type, used for query and programmatic operations

Converting strings with parse:

PARSE is new function in SQL Server 2012 that converts strings to date, time, and number types

sPARSE element	Comment
String_value	Formatted nvarchar(4000) input
Data_type	Requested data type ouput
Culture	Optional string in .NET culture form: en-US, es-ES, ar-SA, etc.

Cast, convert, parse
try_cast, try_convert, try_parse

Cast is standard
convert is not standard
parse is standard or not??

```
SELECT PARSE('02/12/2012' AS datetime2 USING 'en-US') AS parse_result;

--this is the iso format as well as the ansi format. yyyy-mm-dd
select OrderDate from Sales.SalesOrderHeader

select PARSE('2011-05-31' as datetime using 'en-US')
select try_parse('hejc' as datetime using 'en-US')
```

wrong. ISO format is yyyymmdd

Parse is going from string to date and Format will go form date to string

```
SELECT FORMAT(ORDERDATE,'yyyy:MM:dd HH','sv-SE' )FROM SALES.SALESORDERHEADER
```

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

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	Actual length +2 bytes 2* (Actual length) +2 bytes
VARCHAR(MAX), NVARCHAR(MAX)	1-2^31-1 characters	Actual length + 2

CHAR, NCHAR are fixed length

VARCHAR, NVARCHAR are variable length

Character data is delimited with single quotes

use concat. It is standard of handles null

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

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.

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)

IN, LIKE, EXISTS, BETWEEN (predicates)

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

Performance: If the wild card '%' is after the string, then the index on Name column would be used for searching and thus would be fast. If the wild card is at the start of the string, then index could not be used and a scan would have to be performed and would be slow.

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'

```
--datettime2 works correctly due to precision/accuracy...
--datetime would round up here...anyway, u would be off by anwhere between 3 to 1/3 milliseconds and tha
could play up in
--a unexpected way.So .997 is taken as the end of the day
select CAST('2013-09-13 23:59:59.999' as datetime) ;--returns 2013-09-14 00:00:00.000
select CAST('2013-09-13 23:59:59.998' as datetime) ;--returns 2013-09-13 23:59:59.997
select CAST('2013-09-13 23:59:59.997' as datetime) ;--returns 2013-09-13 23:59:59.997
select CAST('2013-09-13 23:59:59.99' as datetime) ;--returns 2013-09-13 23:59:59.990
select CAST('2013-09-13 23:59:59.999' as datetime2) ;--returns 2013-09-13 23:59:59.9990000
```

we should use < tomorrow in datetime predicates to avoid problems like these & not BETWEEN

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

SQL Server doesn't offer an option for entering a date or time value explicitly

- Dates and times are entered as character literals and converted explicitly or implicitly
  - For example, CHAR converted to DATETIME due to precedence
- Formats are language-dependent, can cause confusion

Best practices:

- Use character strings to express date and time values
- Use language-neutral formats → YYYYmmdd

```
--this works
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate = '20110531';

--this works as well
SELECT SalesOrderID, CustomerID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate = '2011-05-31';
```

→ language neutral

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';
```

↓ AS

```
SELECT @DateOnly;--returns 2012-02-12 00:00:00.000
```

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 = '20110531';
```

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 >= '20110531' and OrderDate< '20110601' ;
```

Date and time functions:

Function	Return Type	Remarks
GETDATE()	datetime	Current date and time. No time zone offset.
GETUTCDATE()	datetime	Current date and time in UTC.
CURRENT_TIMESTAMP standard	datetime	Current date and time. No time zone offset. ANSI standard.
SYSDATETIME()	datetime2	Current date and time. No time zone offset
STSUTCDATETIME()	datetime2	Current date and time in UTC.
SYSDATETIMEOFFSET()	datetimeoffset	Current date and time. Includes time zone offset

So should I use current_timestamp over the sysdatetime()

```
SELECT CURRENT_TIMESTAMP;
SELECT SYSUTCDATETIME();
```

Functions that return part of date and times:

Function	Syntax	Return Type	Remarks
DATENAME()	DATENAME(datepart, date)	nvarchar	Use 'year', 'month', 'day' as datepart
DATEPART()	DATEPART(datepart, date)	int	Use 'year', 'month', 'day' as datepart
DAY()	DAY(datevalue)	int	
MONTH()	MONTH(datevalue)	int	
YEAR()	YEAR(datevalue)	int	

```
SELECT DATENAME(year,'20120212');
SELECT DAY('20120212');
```

Functions that return date and time from parts:

Function	Syntax	Return Type
DATEFROMPARTS()	DATEFROMPARTS(year, month, day)	date
DATETIMEFROMPARTS()	DATETIMEFROMPARTS(year, month, day, hour, minute, seconds, milliseconds)	datetime
DATETIME2FROMPARTS()	DATETIME2FROMPARTS(year, month, day, hour, minute, seconds, fractions, precision)	Datetime2
DATETIMEOFFSETFROMPARTS()	DATETIMEOFFSETFROMPARTS(year, month, day, hour, minute, seconds, fractions, hour_offset, minute_offset, precision)	datetime