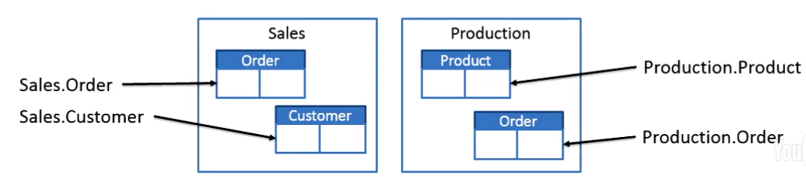
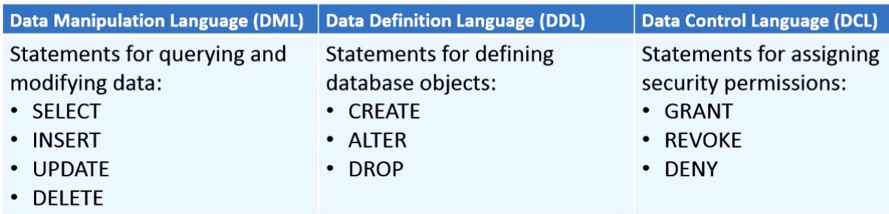
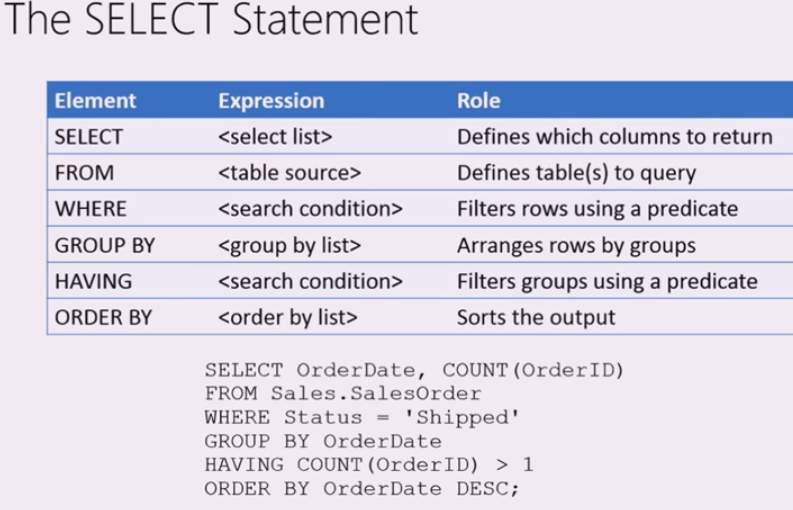
Set based, Entities(tables), row is a instance of that entity, primary key uniquely identifies a row(entity instance) in the table, schema is namespace for db objects like tables, [srvr name][db name][schema name][object name]. two different schemas can have tables with same name

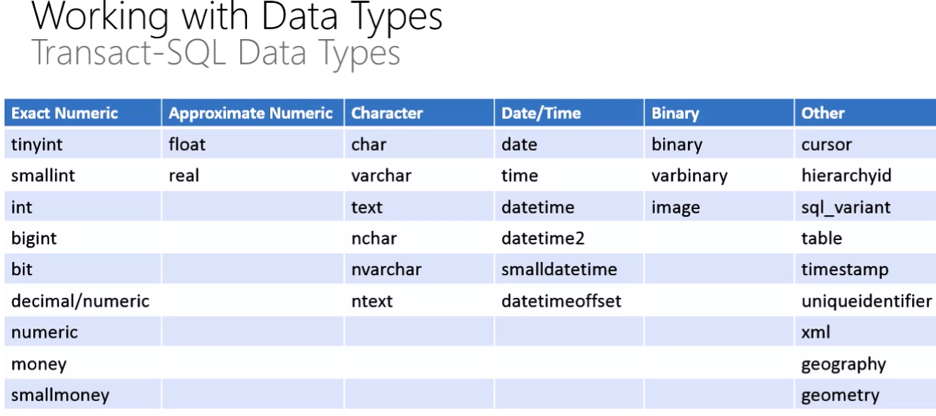




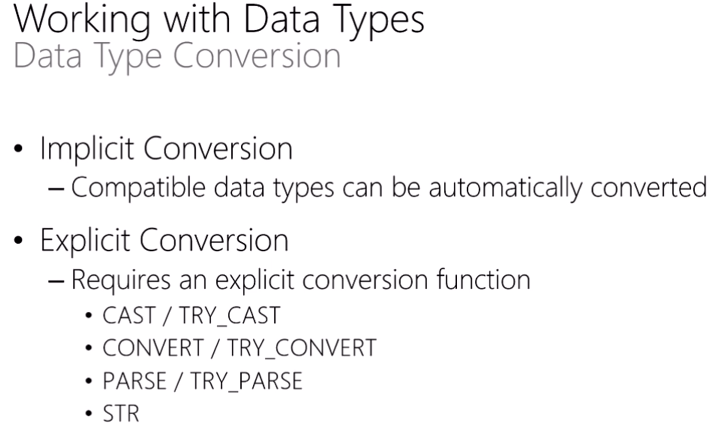
Select total number of orders grouped by order date where there is more than order.



Specify the cols u want explicitly instead of \* to reduce network traffic



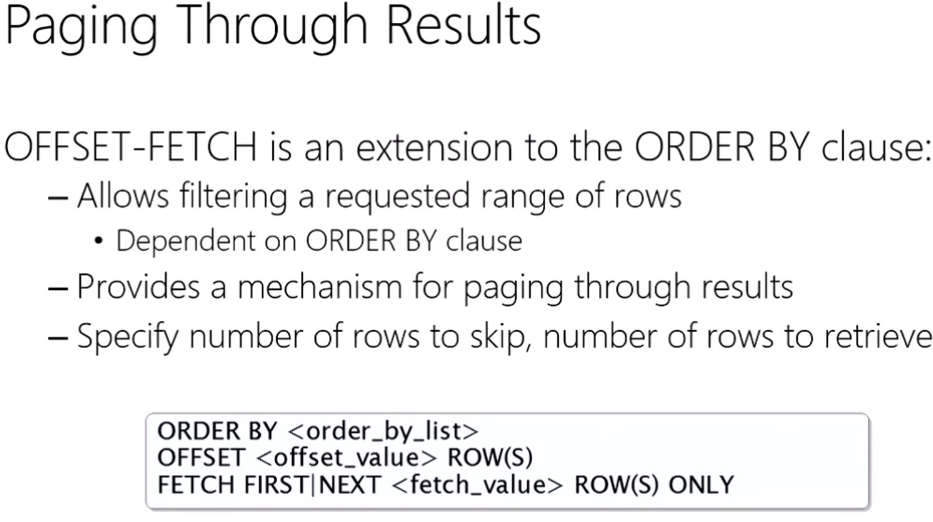
Char and tinyint uses 1 bytes. Nchar uses 2 bytes(equal to wchar in c++)



Cast and convert are pretty much the same. Prefer convert as it has more options for date conversion. parse and str are opposites. Nulls mean unknown or missing value but u can change the default in sql server to get them to mean 0 or empty space. Name is null, isnull (name, value), coalesce selects first not null from a list, case has two forms: one in which the col is outside the when statements and other when it is inside the when statements.

* NULL is used to indicate an unknown or missing value. NULL is **not** equivalent to zero or an empty string.
* Arithmetic or string concatenation operations involving one or more NULL operands return NULL. For example, 12 + NULL = NULL.
* If you need to compare a value to NULL, use the **IS** operator instead of the **=** operator.
* The **ISNULL** function returns a specified alternative value for NULL columns and variables.
* The **NULLIF** function returns NULL when a column or variable contains a specified value.
* The **COALESCE** function returns the first non-NULL value in a specified list of columns or variables).

I think the top function in select statement runs after the order by clause…So order by is not exactly the last thing to happen in all circumstances.???



--THESE QUERIES ARE SAME

SELECT

Color

FROM Production.Product

WHERE Color IS NOT NULL

GROUP BY Color;

SELECT

DISTINCT Color

FROM Production.Product

WHERE Color IS NOT NULL

--THESE QUERIES ARE SAME ARE NOT SAME!!

SELECT

COUNT(Color)

FROM Production.Product

WHERE Color IS NOT NULL

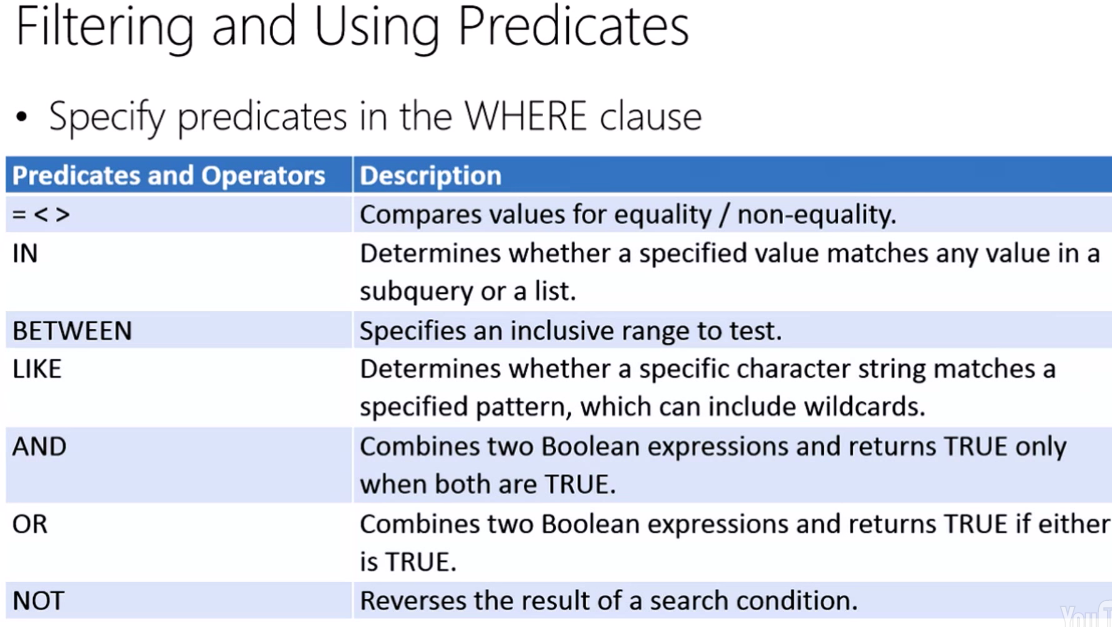
GROUP BY Color;

SELECT

COUNT(DISTINCT Color)

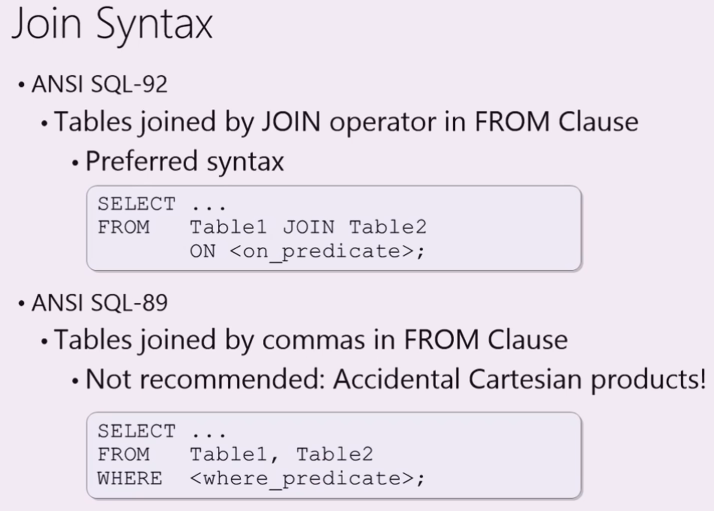
FROM Production.Product

WHERE Color IS NOT NULL



We could use IN once or OR multiple times for the same logic. Between in inclusive but could be replaced with <= and >= which is more readable.

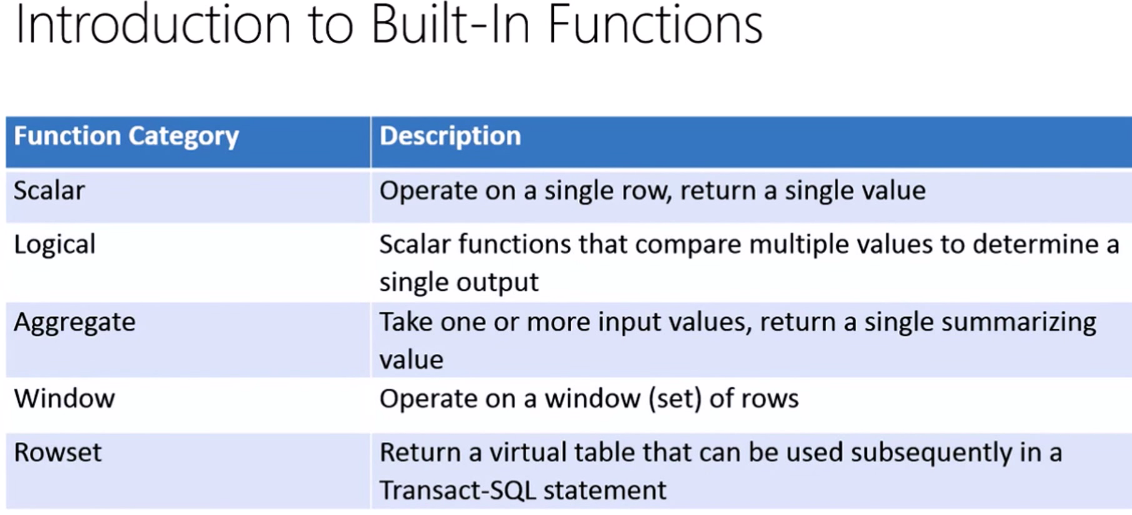
Database can be configured to be case-sensitive or not case-sensitive.



* Joins are used to match rows in one table to rows in another table.
* The query engine supports two ways to define joins: the ANSI SQL-92 syntax (in which the join is specified in the FROM clause) and the older ANSI SQL-89 syntax (in which the join is specified in the WHERE clause). The ANSI SQL-92 syntax is the preferred approach.
* Use a Left Outer Join to include all rows from the first table and values from matched rows in the second table. Columns in the second table for which no matching rows exist are populated with NULLs.
* Use a Right Outer Join to include all rows from the second table and values from matched rows in the first table. Columns in the first table for which no matching rows exist are populated with NULLs.
* Use a Full Outer Join to include all rows from the first and second tables. Columns in the either table for which no matching rows exist are populated with NULLs.

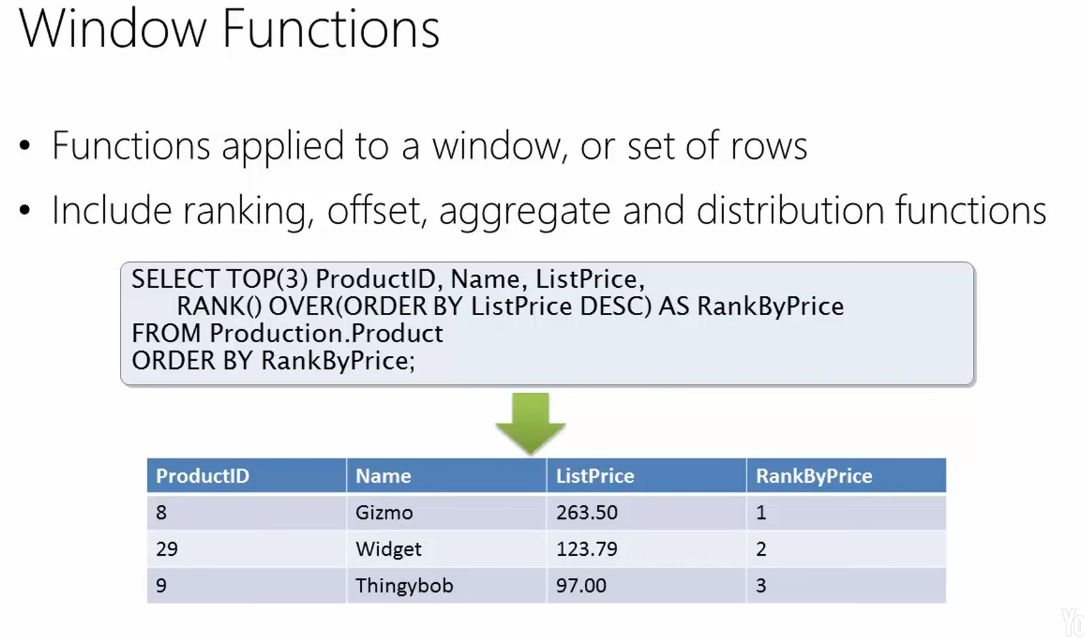
What is the difference between cross joins and full outer joins? Full outer join is the logical foundation on which inner and outer joins are built. So, it happens every time we use a join.

When I have a query, a union and another query, I can define alias only in my first query. The number of cols should be same in unions and in same order I think. And cols must be compatible for implicit conversion if they are not the same at a particular position in the select clause. When we are doing a Union (not ALL), it is as if we are using a distinct clause as it removes duplicates. As union removes duplicates, it has a performance penalty as compared to using Union all.



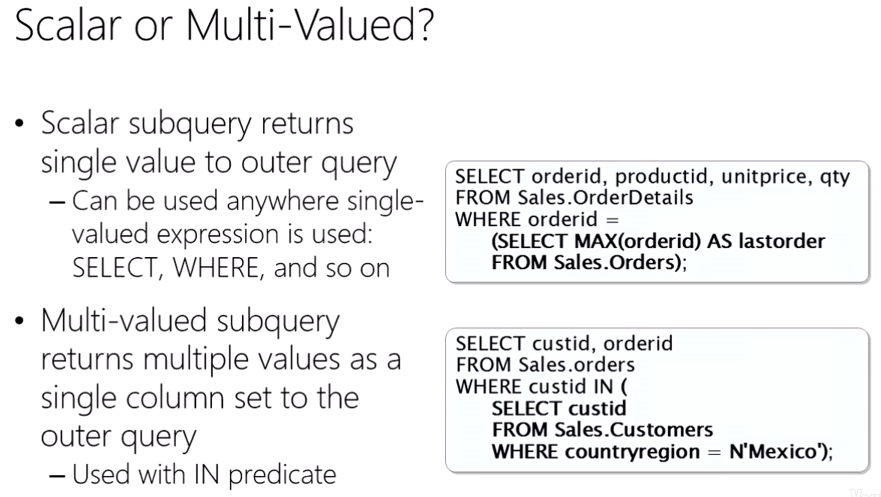
Probably a good idea to test to do case insensitive comparisons, say, by using upper () while comparing with strings.

Rank and order by relation:

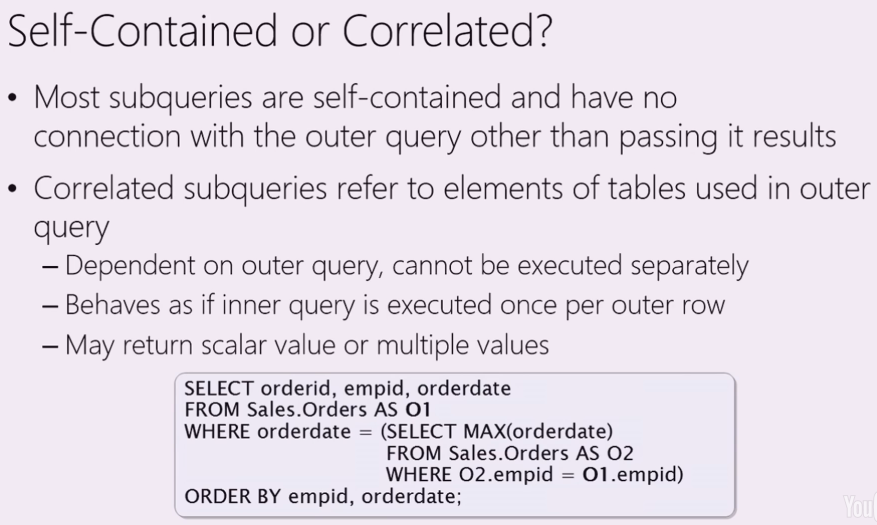


Partition by is like group by.

* Scalar functions return a single value based on zero or more input parameters.
* Logical functions return Boolean values (true or false) based on an expression or column value.
* Window functions are used to rank rows across partitions or "windows". Window functions include RANK, DENSE\_RANK, NTILE, and ROW\_NUMBER.
* Aggregate functions are used to provide summary values for mulitple rows - for example, the total cost of products or the maximum number of items in an order. Commonly used aggregate functions include SUM, COUNT, MIN, MAX, and AVG.
* You can use GROUP BY with aggregate functions to return aggregations grouped by one or more columns or expressions.
* All columns in the SELECT clause that are not aggregate function expressions must be included in a GROUP BY clause.
* The order in which columns or expressions are listed in the GROUP BY clause determines the grouping hierarchy.
* You can filter the groups that are included in the query results by specifying a HAVING clause.



* Subqueries are Transact-SQL queries nested within an outer query.
* Scalar subqueries return a single value.
* Multi-valued subqueries return a single-column rowset.



Correlated sub-query is same as apply.

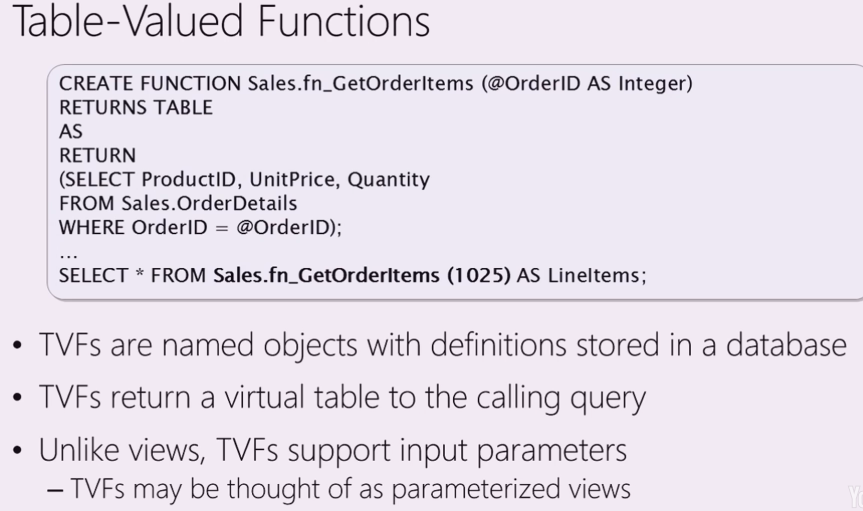
* The APPLY operator enables you to execute a table-valued function for each row in a rowset returned by a SELECT statement. Conceptually, this approach is similar to a correlated subquery.
* CROSS APPLY returns matching rows, similar to an inner join. OUTER APPLY returns all rows in the original SELECT query results with NULL values for rows where no match was found.

Temporary tables:

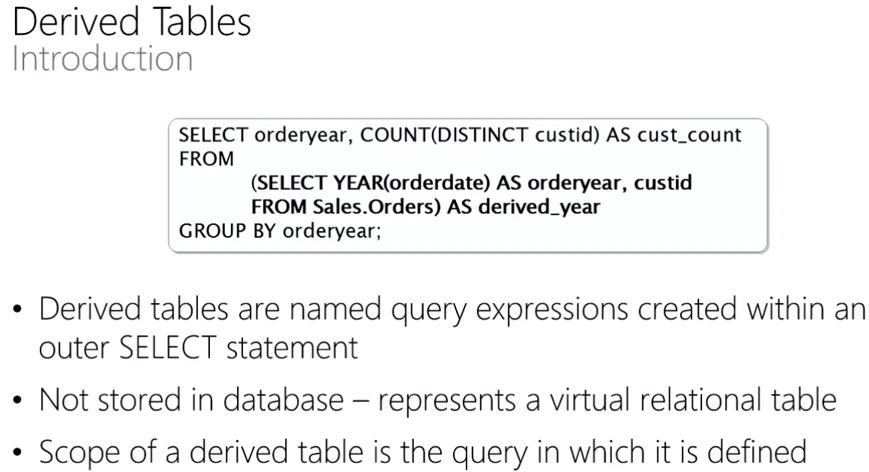


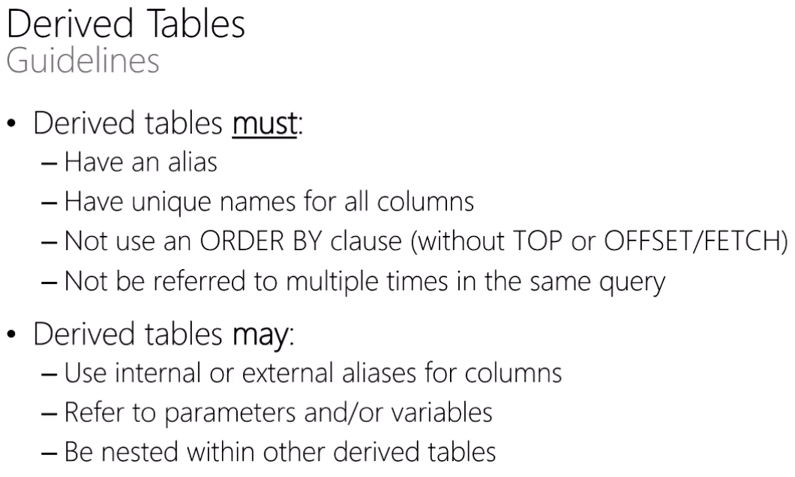
Performance issues with temporary tables. If used inside stored procs, since they are dropped and re-created every time, the stored proc will be recompiled for every call. Might then want to use table variables. Temp table variables works only in batches as they are scoped to a batch.

* Temporary tables are prefixed with a **#** symbol (You can also create global temporary tables that can be accessed by other processes by prefixing the name with **##**)
* Local temporary tables are automatically deleted when the session in which they were created ends. Global temporary tables are deleted when the last user sessions referencing them is closed.
* Table variables are prefixed with a **@** symbol.
* Table variables are scoped to the batch in which they are created.



* Table-Valued Functions (TVFs) are functions that return a rowset.
* TVFs can be parameterized.





* A derived table is a subquery that generates a multicolumn rowset. You must use the AS clause to define an alias for a derived query.
* Common Table Expressions (CTEs) provide a more intuitive syntax or defining rowsets than derived tables, and can be used mulitple times in the same query.
* You can use CTEs to define recursive queries.