**Avoid ORDER BY in SQL Server views**

By: [Aaron Bertrand](https://www.mssqltips.com/sqlserverauthor/49/aaron-bertrand/)

FROM: <https://www.mssqltips.com/sqlservertip/4488/avoid-order-by-in-sql-server-views/>

**Problem**

For as long as I have been working with SQL Server, I have watched people add ORDER BY to views for various reasons - most commonly, they are referencing the view in multiple queries, and want to avoid having to repeat an ORDER BY clause in each of those outer queries.

**Solution**

ORDER BY in a view is not something you can rely on. In fact, it's not even something you can accomplish on its own. Let's look at a few examples using the new [WideWorldImporters sample database](https://blogs.technet.microsoft.com/dataplatforminsider/2016/06/09/wideworldimporters-the-new-sql-server-sample-database):

USE WideWorldImporters;

GO

CREATE VIEW dbo.CustomersByName

AS

SELECT CustomerID, CustomerName, DeliveryCityID

FROM Sales.Customers

ORDER BY CustomerName;

GO

This results in a very explicit error message:

Msg 1033, Level 15, State 1, Procedure CustomersByName  
The ORDER BY clause is invalid in views, inline functions, derived tables, subqueries, and common table expressions, unless TOP, OFFSET or FOR XML is also specified.

Sure, there are kludges to get it in there, most of which I see in the wild are \*exactly\* this:

SELECT TOP (100) PERCENT <columns>

FROM dbo.<table>

ORDER BY <column>;

If we do that with our query above:

CREATE VIEW dbo.CustomersByName

AS

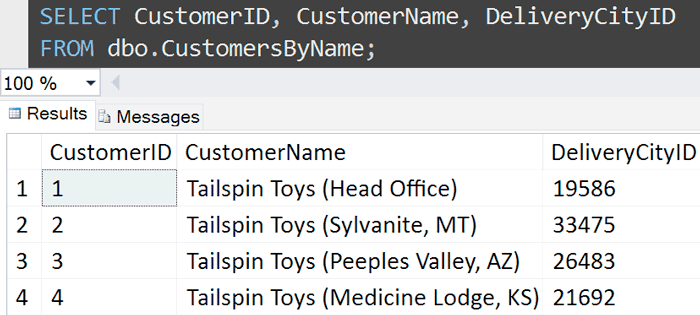
SELECT TOP (100) PERCENT CustomerID, CustomerName, DeliveryCityID

FROM Sales.Customers

ORDER BY CustomerName;

GO

The view is successfully created, but we can easily see that an outer query against the view, without an ORDER BY, won't obey the ORDER BY from inside the view:



This used to work, back in the SQL Server 2000 days, but not in modern versions. Let's compare the execution plans between a query against the view (which returns data sorted by CustomerID) and running the query \*inside\* the view directly (which returns data sorted by CustomerName):

SELECT CustomerID, CustomerName, DeliveryCityID

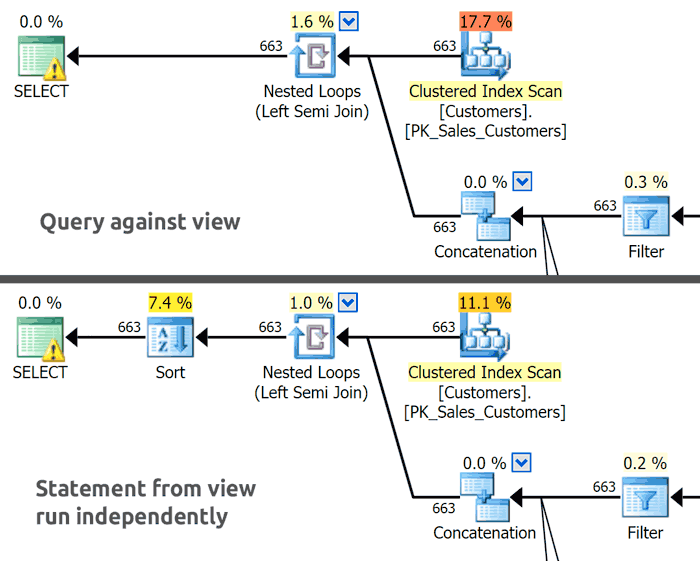
FROM dbo.CustomersByName;

SELECT TOP (100) PERCENT CustomerID, CustomerName, DeliveryCityID

FROM Sales.Customers

ORDER BY CustomerName;

The execution plans are virtually identical, except for one little thing (see if you can spot it):



When we run the query from inside the view (without actually referencing the view itself), the results are ordered as we desired, and this is facilitated by that extra Sort operator. This doesn't happen when we query against the view because, essentially, SQL Server looks at our outer query, sees there is no ORDER BY, and says, "they don't care about the order of results," so feels free to return the data in the most efficient order (which happens to be by CustomerID in this example, but that won't always be true, depending on the columns in the query, other indexes, and other factors).

Since the ORDER BY inside the view is only allowed to exist as a way to determine which TOP rows to include, and since TOP (100) PERCENT means "return all the rows," these are two pieces of logic SQL Server feels confident about completely throwing away and not considering at all. This is why there is no Sort operator in the first plan, and no Top operator in \*either\* plan.

The underlying problem here is that ORDER BY can serve these two functions - one to determine TOP inclusion, and one to determine presentation order. There is no way to prioritize these; in the second query above, the ORDER BY is actually serving both functions. Ideally, SQL Server's TOP clause should have been implemented with its own ORDER BY, so that if we wanted to return the first 10 customers alphabetically but have the results sorted by DeliveryCityID, we could say something like this:

SELECT TOP (10) OVER (ORDER BY CustomerName)

CustomerID, CustomerName, DeliveryCityID

FROM Sales.Customers

ORDER BY DeliveryCityID;

Instead, we have to write our queries a little more elaborately, where we have to perform the two different functions of ORDER BY in two different stages:

SELECT CustomerID, CustomerName, DeliveryCityID

FROM

(

SELECT TOP (10)

CustomerID, CustomerName, DeliveryCityID

FROM Sales.Customers

ORDER BY CustomerName

) AS x

ORDER BY DeliveryCityID;

*(A more common example of this is when you want to present, say, a top 10 list ordered from 10 to 1.)*

[OFFSET/FETCH syntax](https://www.mssqltips.com/sqlservertip/2362/overview-of-offset-and-fetch-feature-of-sql-server-2012/) was introduced in SQL Server 2012, which seems like it could help to solve the problem, but unfortunately it still uses the same overloaded ORDER BY clause.

**Are there other kludges to get the "desired" behavior?**

Of course. There are always going to be ways to get SQL Server to bend to your will, even if only temporarily - by using a percentage so close to 100 percent that it will round up, or by using the upper bound of the BIGINT type:

CREATE VIEW dbo.CustomersByName\_KludgeOne

AS

SELECT TOP (99.9999999999999999) PERCENT CustomerID, CustomerName, DeliveryCityID

FROM Sales.Customers

ORDER BY CustomerName;

GO

CREATE VIEW dbo.CustomersByName\_KludgeTwo

AS

SELECT TOP (9223372036854775807) CustomerID, CustomerName, DeliveryCityID

FROM Sales.Customers

ORDER BY CustomerName;

GO

Sure enough, if you select from each view without an ORDER BY, the results are ordered by name, and the Sort operator is re-introduced into the plan. You'll also notice a new Top operator that wasn't present before, even though no rows are filtered out in either case.

But **please** don't learn from this or rely on it; like the original functionality that was phased away starting in SQL Server 2005, there is always a chance that some future build of SQL Server will "fix" this loophole, and it will stop behaving as you observe. Never mind that even when you \*can\* get the view to return data in a certain order, this may not be the order desired by all queries that reference the view (whether they do so explicitly or not). SQL Server will not be able to determine how to resolve conflicts or prioritization issues with outer queries that have their own, different, ORDER BY clause. You could guess that the outer ORDER BY will always win, but this isn't documented, so is just as unreliable as the other behavior.

Summary

Views are not meant to dictate presentation order; if you expect queries against a view from presenting the data in a predictable order, please stop looking for kludges and add those ORDER BY clauses to your outer queries. Having ORDER BY inside the view is not going to work in all cases, and is going to be confusing for people reading or maintaining the code.

**Next Steps**

See these tips and other resources:

* [Trick to Optimize TOP Clause in SQL Server](https://www.mssqltips.com/sqlservertip/2053/trick-to-optimize-top-clause-in-sql-server/)
* [Overview of OFFSET and FETCH Feature of SQL Server 2012](https://www.mssqltips.com/sqlservertip/2362/overview-of-offset-and-fetch-feature-of-sql-server-2012/)
* [Pagination with OFFSET / FETCH : A better way](http://sqlperformance.com/2015/01/t-sql-queries/pagination-with-offset-fetch)
* [Comparing performance for different SQL Server paging methods](https://www.mssqltips.com/sqlservertip/2696/comparing-performance-for-different-sql-server-paging-methods/)
* [TOP (Transact-SQL)](https://msdn.microsoft.com/en-us/library/ms189463.aspx) (MSDN)
* [ORDER BY Clause (Transact-SQL)](https://msdn.microsoft.com/en-us/library/ms188385.aspx) (MSDN)

Last Update: 10/4/2016