**Advanced Use of the SQL Server ORDER BY Clause**

By: [Jeffrey Yao](https://www.mssqltips.com/sqlserverauthor/121/jeffrey-yao/)

FROM: <https://www.mssqltips.com/sqlservertip/4718/advanced-use-of-the-sql-server-order-by-clause/>

**Problem**

The SQL Sever [ORDER BY](https://msdn.microsoft.com/en-us/library/ms188385.aspx#Case) clause is commonly used in T-SQL scripts, if used properly, it can solve lots of problems. Such as:

* How can I sample [N] records out of a table randomly?
* How can I find the [median](https://en.wikipedia.org/wiki/Median) value of a numeric column?
* How can I order a product table by always putting a specific category at the front of the result?
* How can I find the biggest value records based on a column in each group?

Each question may be resolved multiple ways, but I will focus on using ORDER BY in a simple SELECT statement.

**Solution**

[ORDER BY](https://msdn.microsoft.com/en-us/library/ms188385.aspx#Case) has seen its biggest change in SQL Server 2012 when OFFSET and FETCH key words are introduced. But before that, ORDER BY can still do lots of interesting things. We will look at a few examples here.

Example 1

I have a big table, and I just want to sample 10 records **randomly** for my testing purpose, how can I do it?

We will prepare a sample table and then code against it.

use tempdb

-- prepare test table

if object\_id('dbo.product', 'U') is not null

drop table dbo.product;

go

create table dbo.product (id int identity primary key

, [name] varchar(100)

, [color] varchar(20)

, [price] int

)

go

-- populate the table with some data,

-- there are three distinct colors for each product

; with L0 as (select 1 as c union all select 1 as c)

, L1 as (select L0.c from L0 cross join L0 as T )

, L2 as (select L1.c from L1 cross join L1 as T )

, L3 as (select L2.c from L2 cross join L2 as T )

, L4 as (select L3.c from L3 cross join L3 as T )

, L5 as (select rn = ROW\_NUMBER() over (order by (select null) ) from L4)

insert into dbo.product (name, color, price)

select [name]='Product' + cast(rn as varchar(5))

, color = case ( cast (ceiling(rand(rn\*7)\*100000) as int) %3) when 0 then 'red' when 1 then 'green' else 'blue' end

, price = cast(ceiling(rand(rn\*7)\*123456) as int)%56789

from L5;

Now let's list 10 records randomly.

-- to list top 10 records randomly, this query is good for medium to large (5K) data set.

-- for small data set, using the 2nd way

select top 10 \* from dbo.product

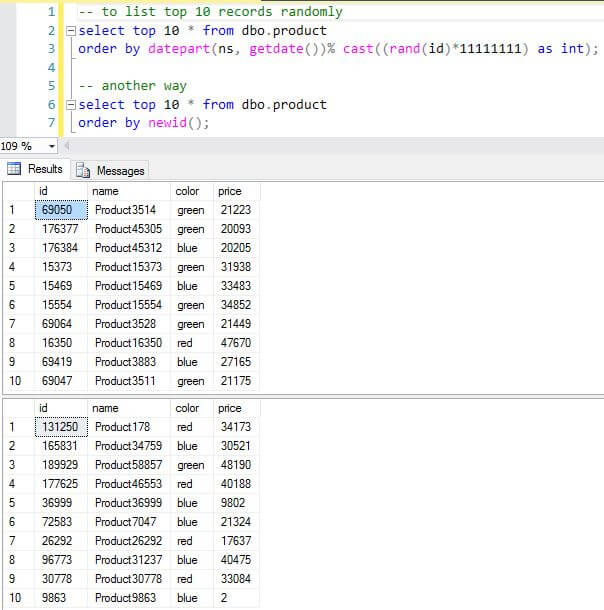
order by datepart(ns, getdate())% cast((rand(id)\*11111111) as int);

-- another way commonly known

select top 10 \* from dbo.product

order by newid();

Each run will return different results as shown below.



Actually there is a third way using OFFSET and FETCH as shown below. This approach will always randomly grab 10 records in sequence.

select \* from dbo.product

order by (select null)

offset cast(ceiling(rand()\*1234567) as int)% (select count(\*) from dbo.product) rows

fetch next 10 rows only

Question 2

Using the same table, we will create a small sample data and then find the median value of [Price].

We will populate the table with 9 records and later with 10 records to see whether our solution works.

truncate table dbo.product

declare @i int = 1;

while @i < 10

begin

insert into dbo.product (name, color, price)

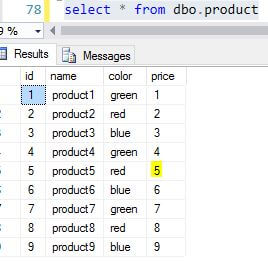
select 'product'+cast(@i as varchar(3)), case @i%3 when 0 then 'blue' when 1 then 'green' else 'red' end, @i

set @i += 1;

end

select \* from dbo.product

The results are shown below:



We can see the median value is 5 for [Price]. So to calculate this in T-SQL, we can use the following code:

-- find the row(s) with median value

select \*

from dbo.product

order by price

offset (select count(\*)-1 from dbo.product)/2 rows

fetch next (case (select count(\*) from dbo.product) % 2 when 1 then 1 else 2 end) rows only

go

-- find the real median value for [Price] column

with c as (

select \*

from dbo.product

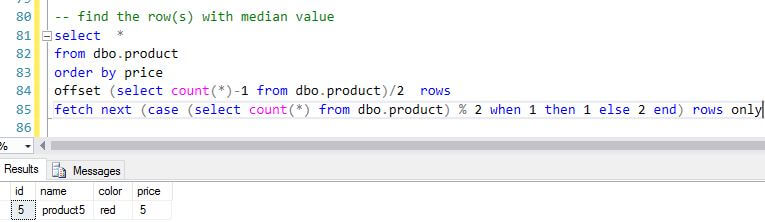
order by price

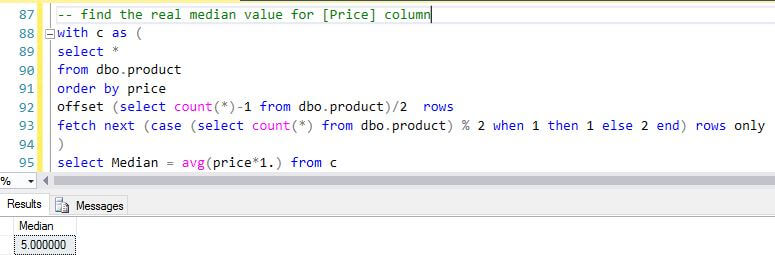
offset (select count(\*)-1 from dbo.product)/2 rows

fetch next (case (select count(\*) from dbo.product) % 2 when 1 then 1 else 2 end) rows only

)

select median = avg(price\*1.) from c





Assume we insert another record into the table to make it 10 records:

truncate table dbo.product

declare @i int = 1;

while @i < 11 -- change 10 to 11

begin

insert into dbo.product (name, color, price)

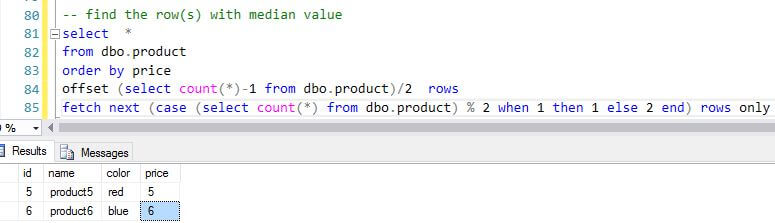
select 'product'+cast(@i as varchar(3)), case @i%3 when 0 then 'blue' when 1 then 'green' else 'red' end, @i

set @i += 1;

end

select \* from dbo.product

The result will be like the following, we will see two median value rows just as expected:



The key here is the OFFSET and FETCH setting, which are decided by the row count, if row count is an odd number, we set the OFFSET to (count-1)/2 and then FETCH next 1 row. But if row count is an even number, we set the same OFFSET value while set FETCH for the next 2 rows.

For an example, if row count = 9, OFFSET = (9-1)/2 = 4, FETCH = 1. If row count=10, OFFSET=(10-1)/2 = 4 (4.5 becomes 4 due to integer conversion), and FETCH=2

Question 3

In the test table created above, there are three distinct colors, Red, Green, Blue, now for my report, I want Green color product to be always on the top of my report, how can I do so in one T-SQL statement?

First let's prepare the sample data, we will still use the same table as created in Question 1, but we will insert 10 records for a better visualization.

-- populate the test table with 10 records only

truncate table dbo.product;

; with L0 as (select 1 as c union all select 1 as c)

, L1 as (select L0.c from L0 cross join L0 T)

, L2 as (select L1.c from L1 cross join L1 T)

, L3 as (select rn = ROW\_NUMBER() over (order by (select null) ) from L2)

insert into dbo.product (name, color, price)

select [name]='Product' + cast(rn as varchar(5))

, color = case ( cast (ceiling(rand(rn\*7)\*100000) as int) %3) when 0 then 'red' when 1 then 'green' else 'blue' end

, price = cast(ceiling(rand(rn\*7)\*123456) as int)%100

from L3

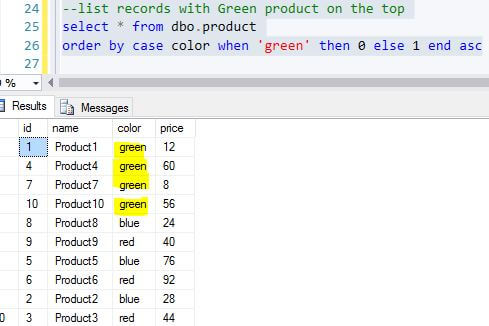
where rn <=10;

--list records with Green product on the top

select \* from dbo.product

order by case color when 'green' then 0 else 1 end asc

The results are shown below:



Notice color Blue and Red are randomly displayed. So another extension to the question could be how to list records in the sequence of Green, Red and Blue

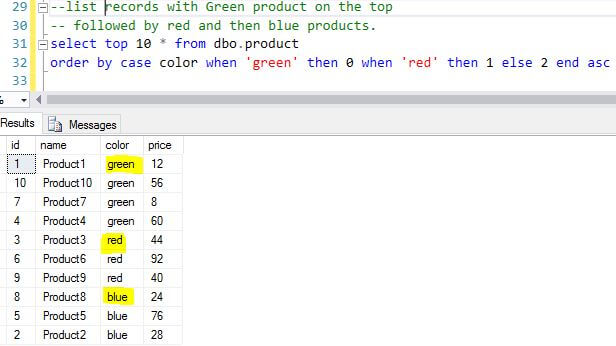
--list records with Green product on the top

select \* from dbo.product

order by case color when 'green' then 0 when 'red' then 1 else 2 end asc

In this ORDER BY clause, we can assign the [color] value to a value, such as [green] to 0, [red] to 1. Since the ORDER BY is defined in ASC order, so when ORDER BY is evaluated, Green will be listed before Red because 0 is less than 1 with ASC order.

The results are shown below:



Question 4

How can I list the most expensive i.e. [price], product in each color group?

We will switch back to the first big sample of data to do the demo, we can just re-run the first table population script to reset up the data.

-- find the most expensive product in each color group.

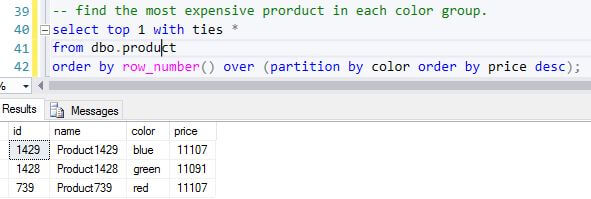
select top 1 with ties \*

from dbo.product

order by row\_number() over (partition by color order by price desc);

ROW\_NUMBER() will list the records in each color group, so the 1st record of each group will have the same value of 1 and they will be automatically selected via **TOP 1 WITH TIES**. When I first used ORDER BY this way long ago, I was totally amazed by the powerful beauty of ORDER BY clause.

The results are shown below:



Sometimes, there can be a few products that are same cost, in this case, instead of using row\_number(), we should use dense\_rank().

-- if there are product that are tied in price in the same group,

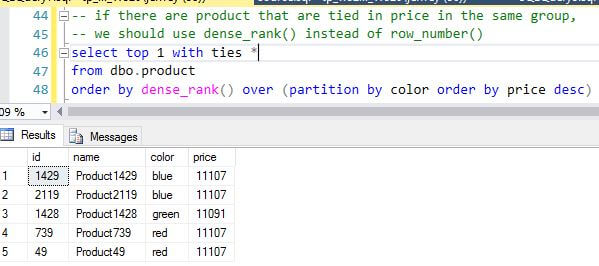
-- we should use dense\_rank() instead of row\_number()

select top 1 with ties \*

from dbo.product

order by dense\_rank() over (partition by color order by price desc)

The results are shown below:



To understand this behavior, we should know the difference of [ROW\_Number](https://msdn.microsoft.com/en-us/library/ms186734.aspx) and [Dense\_Rank](https://msdn.microsoft.com/en-us/library/ms173825.aspx), the ROW\_NUMBER numbers all rows sequentially (for example 1, 2, 3, 4, 5). While DENSE\_RANK provides the same numeric value for ties, the rank of a row is one plus the number of distinct ranks that come before the row in question.

In other words, if two values are the same and are both the biggest value, then both rows will have a rank number = 1, while in row\_number scenario, one row has row number =1 while another will be 2, thus only the row with 1 will be selected by **TOP 1 WITH TIES**.

Summary

In this tip, we have examined some interesting uses of ORDER BY, which are very useful under some niche business requirements. Without ORDER BY, we may write lengthy scripts to achieve the same result.

Currently, OFFSET and FETCH still have some limitations, one of which is that the OVER clause does not support OFFSET and FETCH. If this is supported, we may be able to write more elegant code when doing group processing, such as finding the Nth largest value in each group.

All code is tested with SQL Server 2012 Developer Edition.

**Next Steps**

One item worth digging deeper is to compare the performance of other solutions with the solutions provided in this tip. We can compare their execution plan and the statistics IO/CPU data.

For question 2, if the requirement is to find the median value of each color group, what is your solution?

Also the following links may help to better understand the ORDER BY clause:

* [Different ways to get random data for SQL Server data sampling](https://www.mssqltips.com/sqlservertip/3157/different-ways-to-get-random-data-for-sql-server-data-sampling/)
* [SQL Server Random Sorted Result Set](https://www.mssqltips.com/sqlservertip/3529/sql-server-random-sorted-result-set/)
* [Randomly Retrieve SQL Server Records](https://www.mssqltips.com/sqlservertip/1009/randomly-retrieve-sql-server-records/)

Last Update: 3/15/2017

[**View all my tips**](https://www.mssqltips.com/sqlserverauthor/121/jeffrey-yao/)