[Avoid using constants in an ORDER BY clause](https://www.red-gate.com/hub/product-learning/sql-prompt/avoid-using-constants-in-an-order-by-clause" \o "Avoid using constants in an ORDER BY clause)

FROM: <https://www.red-gate.com/hub/product-learning/sql-prompt/avoid-using-constants-in-an-order-by-clause>

Phil Factor explains why an ORDER BY clause should always specify the sort columns using their names, or aliases, rather than using an integer to specify the position of a column in the SELECT list.

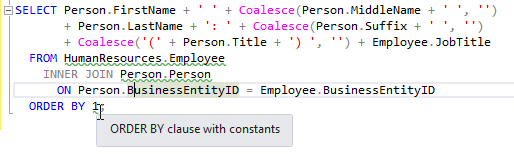
Guest post

This is a guest post from Phil Factor. Phil Factor (real name withheld to protect the guilty), aka Database Mole, has 30 years of experience with database-intensive applications.  
  
Despite having once been shouted at by a furious Bill Gates at an exhibition in the early 1980s, he has remained resolutely anonymous throughout his career.  
  
He is a regular contributor to [**Simple Talk**](http://www.simple-talk.com/) and [**SQLServerCentral**](http://www.sqlservercentral.com/).

Although it is possible to do it, ANSI SQL-92 was the last standard that still supported the use on an integer, the *select\_list\_number*, in the ORDER BY clause, and it was subsequently removed from the ANSI SQL-99 Standard. The major SQL RDBMSs, including SQL Server, still support it, and there is no deprecation notice on it, but it’s on Microsoft’s list of SQL Server features that are on their way out.

Not only is the statement more difficult to understand if you don’t specify the actual column name, but also any subsequent changes to the SELECT list, such as changing the column order or adding new columns, requires a check of the ORDER BY clause, and very likely its modification, in order to avoid unexpected results. Also, use of constants won’t always work, such as when using one to specify an expression that appears in a ranking function.

SQL Prompt’s Best Practice code analysis rule, [BP002 – ORDER BY clause with constants](https://documentation.red-gate.com/codeanalysis/best-practice-rules/bp002), will mean any use of constants in an ORDER BY clause are underlined with the green squiggle.



Why were constants ever allowed in the ORDER BY clause?

Imagine need a list of employee details for your HR boss at AdventureWorks:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | SELECT Person.FirstName + ' ' + Coalesce(Person.MiddleName + ' ', '')         + Person.LastName + ': ' + Coalesce(Person.Suffix + ' ', '')         + Coalesce('(' + Person.Title + ') ', '') + Employee.JobTitle AS employee,    Employee.NationalIDNumber, Employee.BirthDate, Employee.MaritalStatus,    Employee.Gender, Employee.HireDate, Employee.SalariedFlag,    Employee.VacationHours, Employee.SickLeaveHours    FROM HumanResources.Employee      INNER JOIN Person.Person        ON Person.BusinessEntityID = Employee.BusinessEntityID; |

No ordering was specified so the current ordering is a matter of chance. You now realize that you need to order it by the employee’s name:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | SELECT Person.FirstName + ' ' + Coalesce(Person.MiddleName + ' ', '')         + Person.LastName + ': ' + Coalesce(Person.Suffix + ' ', '')         + Coalesce('(' + Person.Title + ') ', '') + Employee.JobTitle AS employee,    Employee.NationalIDNumber, Employee.BirthDate, Employee.MaritalStatus,    Employee.Gender, Employee.HireDate, Employee.SalariedFlag,    Employee.VacationHours, Employee.SickLeaveHours    FROM HumanResources.Employee      INNER JOIN Person.Person        ON Person.BusinessEntityID = Employee.BusinessEntityID    ORDER BY employee; |

Smiles all around. It worked. There was, long ago, a time that it didn’t work. With SQL, at that time, if you wished to order by an expression involving one or more columns, SQL couldn’t do it. This was because there was no obvious way to know the values of the expression in the result, at the point in the execution that the sorting was done. Strangely, also, the SQL Standard at the time required that you could only specify a column in the SELECT statement to be a parameter in the ORDER BY clause.

Using expressions in the ORDER BY clause

SQL then allowed expressions in ORDER BY clauses. If one of the columns was an expression, and you wanted to sort by it, then the expression in the ORDER BY clause had to match exactly the expression used in the SELECT statement of the table source:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | SELECT Person.FirstName + ' ' + Coalesce(Person.MiddleName + ' ', '')         + Person.LastName + ': ' + Coalesce(Person.Suffix + ' ', '')         + Coalesce('(' + Person.Title + ') ', '') + Employee.JobTitle AS employee,    Employee.NationalIDNumber, Employee.BirthDate, Employee.MaritalStatus,    Employee.Gender, Employee.HireDate, Employee.SalariedFlag,    Employee.VacationHours, Employee.SickLeaveHours    FROM HumanResources.Employee      INNER JOIN Person.Person        ON Person.BusinessEntityID = Employee.BusinessEntityID    ORDER BY Person.FirstName + ' ' + Coalesce(Person.MiddleName + ' ', '')         + Person.LastName + ': ' + Coalesce(Person.Suffix + ' ', '')         + Coalesce('(' + Person.Title + ') ', '') + Employee.JobTitle; |

This was not only unwieldy, but if you started fiddling with the column expression and forgot to change the ORDER BY to match it could suddenly cause the query to run slowly, if it then wasn’t identical:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | SELECT Person.FirstName + ' ' + Coalesce(Person.MiddleName + ' ', '')         + Person.LastName + ': ' + Coalesce(Person.Suffix + ' ', '')         + Coalesce('[' + Person.Title + '] ', '') + Employee.JobTitle AS employee,    Employee.NationalIDNumber, Employee.BirthDate, Employee.MaritalStatus,    Employee.Gender, Employee.HireDate, Employee.SalariedFlag,    Employee.VacationHours, Employee.SickLeaveHours    FROM HumanResources.Employee      INNER JOIN Person.Person        ON Person.BusinessEntityID = Employee.BusinessEntityID    ORDER BY Person.FirstName + ' ' + Coalesce(Person.MiddleName + ' ', '')         + Person.LastName + ': ' + Coalesce(Person.Suffix + ' ', '')         + Coalesce('(' + Person.Title + ') ', '') + Employee.JobTitle; |

This version of the query ran 25% slower because the expressions don’t match. It wasn’t a big deal but was an irritation and looked clumsy.

Using constants in the ORDER BY clause

SQL initially got around the problem, without fixing the underlying flaw, by allowing you to specify an integer constant in the ORDER BY expression, corresponding to the column number in the table expression that you were ordering. A sort column in SQL Server can now be specified as a name or column alias, or a positive integer representing the position of the column in the select list. This was unwise because it meant that if you changed the order of columns in the SELECT statement then things went wrong. It also doesn’t work when the ORDER BY expression appears in a ranking function.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | SELECT Person.FirstName + ' ' + Coalesce(Person.MiddleName + ' ', '')         + Person.LastName + ': ' + Coalesce(Person.Suffix + ' ', '')         + Coalesce('(' + Person.Title + ') ', '') + Employee.JobTitle,    Employee.NationalIDNumber, Employee.BirthDate, Employee.MaritalStatus,    Employee.Gender, Employee.HireDate, Employee.SalariedFlag,    Employee.VacationHours, Employee.SickLeaveHours    FROM HumanResources.Employee      INNER JOIN Person.Person        ON Person.BusinessEntityID = Employee.BusinessEntityID    ORDER BY 1; |

Conclusion

Microsoft long ago fixed the original underlying flaw in the SQL Server query optimizer, which was that you were unable to reference the multi-column expression by its alias because, at that point in the optimization process, the values in the alias weren’t known. Now it is all easy, but in the interim a lot of code got written specifying the column number in the ORDER BY clause. It made expressions look neater and saved a few keystrokes. They looked as if they’d run faster but shared the same execution plan as those ordering by the full expression. The quick fix of using integer constants was removed from the SQL Standard before the millennium but lives on in a strange half-life. It is very difficult to remove a feature once it has got established.