T-SQL Coding Standards

SQL isn't rocket science. At the very least, it's safe to say that very little SQL in general use should pose a comprehension barrier to any developer, except for those who are particularly Hard of Understanding.

However, poorly conceived, poorly structured, poorly formatted SQL is rife, and the majority of difficulties in figuring out others' code seem to be due solely to careless coding style.

Organisations typically have development standards in place for database design, VBScript, Java and C#, but for some reason SQL is often overlooked. What follows are some thoughts I put together in my previous team for how things should be done, in the hope of providing a basis for a SQL coding standard, and in anticipation of readers providing their feedback.

Chris Lacey (12 Oct 2004) www.cslacey.co.uk/contact

Capitalise

All SQL keywords and function calls should be capitalised (SELECT, DELETE, FROM, WHERE, CASE, GETDATE()) to allow them to be easily distinguished when viewed outside of Query Analyser, *e.g.* in environments which don't support keyword colouring.

Also, the same capitalisation as appears in the database should be used for all table and column names, as if they were case sensitive: `Customer.CompanyName` or `pub_info.pr_info`. This provides consistency, and also permits SQL Server to cache execution plans for reused statements.

When creating tables and columns, use singular names with PascalCasing (e.g. *EmployeeTerritory* table, NOT *EmployeeTerritories* or *employee_territory*). Treat acronyms as words (e.g. *AbcCorpEmployeeId*, not *ABCCorpEmployeeID*) to assist in readability.

Avoid Aliasing

Use column and table aliasing only where absolutely necessary, and never as a means of reducing keystrokes, *e.g.*

```
SELECT
   Order.OrderId,
   Order.ShippedDate,
   Customer.ContactName,
   Customer.Address
FROM Order
   LEFT JOIN Customer ON Customer.CustomerId = Order.CustomerId
```

and NOT:

```
SELECT
   o.OrderId,
   o.ShippedDate,
   c.ContactName,
   c.Address
FROM Order [o]
   LEFT JOIN Customer [c] ON c.CustomerId = o.CustomerId
```

In this example, *Customer.Address* is immediately meaningful without need for cross-referencing the remainder of the code. This becomes increasingly important as SQL statements get longer. When working with numerous blocks of SQL, it would become easy to muddle up, say *c* the Customer table, with *c* the Categories table.

This also makes for easier copying and pasting of code blocks between queries.

Indent

Indentation should be used for all subqueries and subclauses. Level 0 should contain only the most basic keywords (e.g. SELECT, DELETE, WHERE, ORDER BY, HAVING). Select lists (part of the SELECT clause), join information (part of the FROM clause), and so on, should therefore be indented as illustrated:

```
SELECT
   Product.ProductId,
   Product.ProductName,
   Product.UnitsInStock,
   Supplier.City,
   Category.Description
FROM Product
   LEFT JOIN Supplier ON Supplier.SupplierId = Product.SupplierId
   LEFT JOIN Category ON Category.CategoryId = Product.CategoryId
WHERE Product.UnitsInStock > 10
   AND Supplier.City = 'London'
   AND Category.Description IS NOT NULL
ORDER BY
   Product.UnitsInStock DESC,
   Supplier.City ASC,
   Category.Description ASC
```

Where subqueries are used, the first line of the query should start on a new line, and the whole block should be indented as appropriate. The brackets used to demark the subquery should be on separate lines to permit easy selection of the block as a query in its own right (e.g. for copying and pasting elsewhere). For example:

```
...
WHERE Product.UnitsInStock > 10
   AND Product.ProductId IN (
      SELECT ProductId
      FROM Order
      WHERE CustomerId = 123
   )
```

Comment

Explanatory comments should be embedded within SQL in the same way as they should in any coding language. Use /* Comment */ notation to delineate comments clearly.

In most cases, subqueries should be commented to describe what function they perform, e.g.

```
...
AND Order.RequiredDate < (
   /* Soonest delivery date for top priority product */
   SELECT TOP 1 ExpectedDate
   FROM SupplierDelivery
   WHERE Cancelled = 0
      AND ProductId = (
         /* Highest priority product in the customer order */
         SELECT TOP 1 ProductId
         FROM OrderDetail
         WHERE OrderId = @OrderId
         ORDER BY Priority DESC
      )
   ORDER BY ExpectedDate DESC
   )
```

Use multiple, independent, lines

SQL statements should be divided into multiple lines, except in very basic cases, e.g.

```
SELECT ContactName FROM Customer WHERE ContactName LIKE 'L%'
```

Wherever possible, make each line an "independent" entity which - if removed - would not make the remaining SQL invalid. This makes the query easy to add to and modify over time. For example:

```
SELECT
   Product.ProductId,
   Product.ProductName,
   Product.UnitsInStock,
   Supplier.City,
   Category.Description
FROM Product
   LEFT JOIN Supplier ON Supplier.SupplierId = Product.SupplierId
   LEFT JOIN Category ON Category.CategoryId = Product.CategoryId
WHERE Product.UnitsInStock > 10
   AND Supplier.City = 'London'
   AND Category.Description IS NOT NULL
ORDER BY
   Product.UnitsInStock DESC,
   Supplier.City ASC,
   Category.Description ASC
```

In the above example, by inserting or deleting lines in the correct position:

- Column names in the select list can be added or removed (the last entry obviously having to be a special case in that it has no trailing comma).
- Table joins can be added or removed (ensure the JOIN instruction and ON filter are on the same line)
- WHERE filters can be added or removed (ensure the conjunction appears on the same line as the logical expression)
- ORDER BY terms can be added or removed (last entry again having no trailing comma).

Use SQL Parameters

When embedding values into queries within code, use proper SQL parameters instead of string concatenation, e.g.

```
string sql = "SELECT City FROM Employee WHERE EmployeeId = @EmployeeId";
SqlParameter[] parameters = new SqlParameter[1];
parameters[0] = new SqlParameter("@EmployeeId", SqlDbType.BigInt);
parameters[0].Value = employeeId;
return (string) SqlHelper.ExecuteScalar(
   connectionString, CommandType.Text, sql, parameters);
```

and NOT:

```
string sql = "SELECT City FROM Employee WHERE EmployeeId = " + employeeId;
return (string) SqlHelper.ExecuteScalar(
   connectionString, CommandType.Text, sql);
```

This helps enforce typing, guards against SQL insertion attacks, and permits SQL Server to cache and reuse the execution plan when executing the query with a different value of the parameter.

Allow Block Copy and Paste to/from Query Analyser

In languages which allow this, embed SQL queries into code in such a way as to allow for copying and pasting of the entire block into Query Analyser without need for any manual reformatting, e.g.

```
string sql = @"
   SELECT
      FirstName,
      LastName
   FROM Employee
   WHERE City = 'London'
";
```

and NOT:

```
string sql = "SELECT" +
   "    FirstName," +
   "    LastName" +
   " FROM Employee" +
   " WHERE City = 'London'";
```

This also avoids the need for remembering to insert additional preceding or trailing spaces after each line to form valid SQL.