Best Practises For SQL Server

by Rodney Ellis

With over a decade of writing all sorts of applications that read and write to Microsoft SQL Server - Windows, Web, Apps - with all sizes of databases; I've seen a lot of good databases and an awful lot of terrible stuff.

I have also been fortunate enough to work with some of Australia's best SQL Server gurus, including a lot of time being mentored by Victor Isakov.

Below is my current list of best practises for developing and tuning databases with SQL Server. I hope you find it useful!

Tables

All tables should be created with the following:

- One field created as an IDENTITY column.
  - This should be defined as INT, BIGINT, or in rare cases a SMALLINT.
  - Make it the first column in the table so it’s easy to spot and reference.
  - It must have an index directly relating to it (in most cases it will be the Primary Key and Clustered Index too).
  - (I am aware of a couple of other bloggers saying that they're not necessary on every table ... but in the real world, when you need to update/remove a row, having a unique ID for each row makes it so much easier!

- Field Data Types:
  - Character fields greater than 5 characters should be created as VARCHAR or NVARCHAR.
  - Do not use any deprecated field types (text, ntext etc ...) see: https://msdn.microsoft.com/en-us/library/ms143729.aspx
  - For date and time fields ... combine them with DATETIME2(0), or DATETIME2(7), depending on how much detail you require.
  - If you need only the date, then use DATE.
  - And if you only need the time, then use TIME.

    - Only use FLOAT for very large numbers, otherwise use the smallest possible integer type, or DECIMAL.

- A PRIMARY KEY
    - By definition these are unique values.
    - Try to keep them to only 1, 2, or 3 fields so that SQL doesn't have to do too much work to maintain.
    - Following on from the point above, it is reasonable to use the IDENTITY column if there is nothing too obvious or small to use.

- A CLUSTERED INDEX, which
    - Should be a unique value, but doesn't always have to be as in some cases it makes sense for the table to be sorted over the columns which are commonly retrieved.
    - Same as with the PK above, using the IDENTITY field is good so that SQL has less to keep in order.
    - Do not use over too many fields, and especially not on hard to maintain fields such as GUIDs.

- TRIGGERS should NOT be used
    - If you must use them, then do not alter data in any other table from the trigger.
    - Only use them for auditing, or maintaining small things like timestamps.
    - In case you misread the first point … Triggers should not be used.

Indexes

Keep in mind that every index in the system:

- Takes up its own space on the hard-drive in memory.
- When the underlying fields are changed, the index needs to be changed.

So … do not create indexes on the table unless they're necessary (ie, the data is called in this way frequently).

- Also, if the table has an IDENTITY field on it (see above), then add an INCLUDE with this field to the index.
- Ensure you have routine that runs periodically (for example at night) that Reorganises and/or Rebuilds the indexes.

Stored Procedures

When accessing the database, Stored Procs should be used instead of inline code.

However …

- Try not to put too much logic in there (obviously some is ok). Keep them just to INSERT, UPDATE and DELETE Statements where possible (ie, avoid expensive CURSORS and WHILE loops)

- Try to use UPPER CASE for all T-SQL commands, just like in this document. This makes it far more readable, especially if printed out. And on the same note, keep it formatted nicely with all FROM and JOIN statements starting on a new line so it's easy to see which tables are referenced
- DO NOT use SELECT * … please define all columns
- Use the semicolon ";" at the end of each statement. This is becoming best practice in MS-SQL, and is required with CTEs

Naming:

- DO NOT name them "sp_" as this causes the optimiser to go through all the system stored procs first.
- It's best to name them in relation to the main table, and then the function, so they're easy to find when scrolling down the list. For example:
  - spAccountInsert
  - spAccountDelete
  - spAccountUpdate
- Each of these names gives the new developer a pretty good understanding of what the stored proc does without having to open it.

Functions

Wherever possible, make sure that any functions created do not look at tables. This is because if you put the function into the query, then it's possible it can be doing this work for every row in the query … effectively creating a CURSOR (sometimes the optimiser turns it into a join, and sometimes a cursor … depends on the specifics of the SQL).

Also, be careful when putting the function call in the JOIN clause because this may negate the use of any indexes.

Locking

For the most part, let SQL manage locking. No need to set the ISOLATION LEVEL unless it's necessary. Remember that Microsoft spends countless hours and dollars working on Locking so that you don't have to!

But sometimes, when doing a quick ad-hoc SELECT on a production table, for example a large Logging Table, and you're not sure of the table indexes or if you're going to lock the rows … then it is best to add a WITH (NOLOCK) to the end of the FROM clause for each table, just so you don't cause any unintentional locking.

Cursors

If you have to use a CURSOR (a quick Google search will explain why not to) … then please make sure you use it with FAST_FORWARD and make sure it's DEALLOCATEd once no longer used.