# Really Bad Advice for SQL Server Administrators

I asked many SQL Server administrators for the absolute dumbest advice they have ever been given by real people recently and this (perhaps a bit too humorous) article is the culmination of what I was told. The advice ranges from horrible to the extremely horrible, but none of it could be considered useful whatsoever.

The advice ranges from using the SA password, testing in production, stuffing all your data processing processes onto a small server with limited RAM, to turning on both autogrow and autoshrink to compete with each other. All, I might add, while keeping the amount of memory on your server to as low as possible.

If you feel like you agree with any of this advice, you need to take a look at yourself in the mirror and realize you may not be a very good DBA.

## System testing in production is the best way to know your code truly works

One way to ensure your application is running as desired is to run extensive testing using the most real data possible. Where is the most real data? Production.

By using your production system to fully test all new code, you discover problems with the best test data and the most real load possible. It is essential to run some of your test transactions through your production systems, like purchasing one of your products, but you can see the real users working through any bugs live so they can report them to you.

Note that this doesn't mean you shouldn't do testing on your source code using unit tests or ever test-driven development methodology. It means that once you commit your code and are ready to test the system, production will give you the best results.

## Use as few servers as necessary

Do you need SQL Server Analysis Services, the Relational Engine, SSIS, DQ, and a Web Server? Running them on the same machine saves on hardware and licensing costs and makes it easier for these services to communicate more efficiently.

If you need to scale these services, adding CPUs and disk space (and possibly RAM) is far easier than dealing with multiple servers as usage levels increase.

## Use auto-shrink to keep your data files lean and mean

With disk space costs becoming increasingly expensive, you need to use an SQL Server database setting called AUTO\_SHRINK.

This setting will remove unused space in your database once a day so there are no gaps in the data files. A best practice is to run this command daily using the LEAVE\_SPACE\_FOR\_A\_DAY parameter. This will scan the system event log and determine how much data your users have created in this database on similar days of the month for as long as you have log rows in the log.

## Autogrow files so they are basically self managing

Along with the advice to use auto-shrink in all of your databases, it is essential always to use the complimentary AUTOGROW setting on your database to let the data files grow larger even when the database is shrinking.

## The SA account saves time

Simply put, having one production account with one well-known password saves time in many ways. By sharing this password with any person who needs administrative access to the server, there are fewer passwords and users to manage, so there is less chance that that password will be lost to a bad actor.

Note: It is imperative to have NDAs and training classes on how to keep this password safe. By keeping the password human readable, you are ensured that users will not write this down anywhere (even in a secure password manager)

For example, you don't have to be awoken at night to fix deployments that have broken the database server.

## Indexes on a binary-valued column can be tremendously useful

While an index with unique values is obviously useful, one of the best uses for an index is on a column with just two values. While this may sound counterintuitive, a "binary index," as it is commonly referred to, halves the search space and makes table scans, on average, half of a table. This is an especially useful technique to improve performance for active vs. inactive flags in customer tables.

## Keep your server memory low when your databases are large

There is a myth that database servers require a lot of RAM for processing. While it is true that your data is pulled into memory as it is used. SQL Server uses a stack algorithm to determine what data to remove from memory.

So if you read data A, B, and then C, it will remove C from memory first, so it doesn't matter how much RAM you have; it makes it faster to cache data as you read it, but holding it in RAM is of no value.

Hence, it is important to make sure your RAM is limited. As an example, say your database is 1TB in size, a typical max size of 64 GB of RAM is a good place to start, and then consider taking away some of the RAM during testing.

## ORMs are the best way to develop well performing database code.

Always use the object-first ORM approach to generate your database for optimal and efficient schemas. Avoid database-specific features like table-valued functions or materialized views to prevent vendor lock-in without added benefits.

By letting the ORM "write" the T-SQL code for you, you give the ORM the power to morph the code to the best possible version using the built-in AI systems that ORM systems have had for years.

## Always use NOLOCK in almost each and every SQL statement.

Telling the query processor not to use locking is beneficial for almost any query. Using NOLOCK works for SELECT queries and also for INSERT, UPDATE, and DELETE statements. By doing this, you can turn any database from fast to extremely fast with only the slightest chance of any data integrity issues.

## You shouldn't change the default MAXDOP, as it has been set for you to handle all situations.

Many settings are available to you in SQL Server, and one of the most important to leave alone is MAXDOP. The MAXPOP setting sets the maximum degree of parallelism that a query can use, which is basically the number of processors that the query processor can use to execute your query.

While it is available as a setting, it should typically be left to whatever value is set for the server. This value starts at zero, meaning it can use only a single processor unless it senses the query is being used in a Business Intelligence query, in which case it can use 1 or 2.

## For BI systems, there is little to no need for a backup or restore plan

Since BI systems all use data based on existing systems, there is no need to have any recovery plan other than keeping a copy of the data structures available. If you lose the database, you can simply run all of the ETL pipelines again. It is rare that a database will crash, and most developers and administrators make so few mistakes that the cost of the backup system alone will make the delays in returning to full operation worth it.

This allows you to not only save on the costs of backups but also to get a refresh of the data and verify that all of the code you have written is working as it is expected to.

## SQL Server works better with LOOPING code

If you want to filter from a set of data, the best way is to use a WHILE WEND construct and do the filtering ROW BY ROW. For example, say you have a table with a million rows. You can write:

SELECT \*

FROM TableName

WHERE Criteria = False;

But it is often far faster to use a loop such as:

CREATE TEMP TABLE #Results

WHILE (RowsStillInTable)

BEGIN

INSERT INTO #Results

SELECT TOP 1 \*

FROM TableName

WHERE Criteria = False;

IF @@RowsCount = 0

Break;

WEND;

By using the TOP 1 modifier, this allows each iteration to execute much faster than the SELECT statement that needs to touch every row in the table simultaneously.

## Conclusion

If you recoiled at all of these ideas listed, you have passed my test. If you have ever heard any of this advice in the past from fellow data platform workers, well, that makes you a pretty normal data platform worker.