

SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Discussion: Table Design Strategies

Kimberly L. Tripp
Kimberly@SQLskills.com



Database Development and Design

- **Whose job is it?**
- **Resources**
 - Pluralsight: SQL Server: Why Physical Database Design Matters
 - Author/Presenter: Kimberly L. Tripp, SQLskills.com
 - <http://pluralsight.com/training/Courses/Description/sqlserver-why-physical-db-design-matters>
 - Pluralsight: Developing and Deploying SQL Server ISV Applications
 - Author/Presenter: Erin Stellato, SQLskills.com
 - <http://pluralsight.com/training/Courses/Description/sqlserver-developing-deploying-supporting-isv-applications>
- **Things to consider**
 - Data type best practices
 - Understanding row width (vertical partitioning)
 - Application inconsistencies in types
 - The cost of poor design



Use the “Right” Data Type

System supplied data types:

- Binary
- Character
- Integers
- Exact numerics
- Monetary
- Date and time types
 - Lots of new date/time data types in 2008/2012
- Legacy LOB (image, (n)text)
- LOB (“max” types, XML)
- Uniqueidentifier (GUID)
- FILESTREAM (vs. LOB)

Find the “right” data type for the job:

- * Use the *smallest (but least restrictive) data type possible*
- * Use *Unicode over ASCII*
- * If the data type varies:
 - < 5 chars should be nchar
 - 5-20 chars – questionable
 - > 20 char – lean towards nvarchar
- * For decimal/numeric data:
 - Find the right range
 - Standardize on decimal or numeric
 - Understand precision and range
 - Consider vardecimal in 2005 or row compression in 2008+
- * Use *uniqueidentifier sparingly*
- * Consider “sparse” attribute for 2008+ (for flexible design)

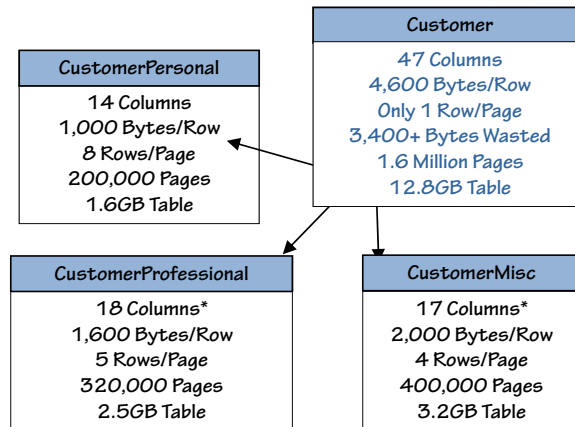
Optimal Row Width

- Consider table usage above all else
- Estimate average row length
 - Overhead
 - Fixed-width columns
 - Estimate average from realistic sample data

```
SELECT avg (datalength (columnname))
FROM tname
```
 - Review min, max and avg. row width of existing and/or sample tables

```
sys.dm_db_index_physical_stats
```
- Calculate page density (rows/page):
 $8,096 \text{ bytes/page divided by } ??? \text{ bytes/row} = \text{rows/page}$
- Calculate wasted bytes – on disk and in memory

Consider a Customer Table With 1,600,000 Rows



Customer = 12.8GB

Partitioned Tables

= 7.3GB

- Savings in overall disk space (5.5GB saved)
- Not reading data into cache when not necessary
- LOB data can be isolated to support online index operations (prior to 2012 where rebuilds with LOB can be done online) to more critical data
- Locks are table specific therefore less contention at the row level

* The PRIMARY KEY column(s) must be made redundant for the additional tables.
 Above: 47 columns in Customer; 49 columns total between 3 tables.

Vertical Partitioning

- **Optimizing row size for:**
 - Caching: better page density means less memory required
 - Locking: only locking the columns that are of interest minimizes even row-level conflicts
- **Usage defines vertical "partitions" or "sets"**
 - Logically group columns to minimize joins
 - Consider read only vs. OLTP columns (LOB separate from OLTP to allow online index maintenance (prior to 2012) for the critical/OLTP part of the table)
 - Consider columns often used together
- **If every query requires a join, this isn't as optimal as it could be but should still be considered**

Pushing LOBs “Out of Row”

- Subtle form of vertical partitioning
- Doesn’t affect the application
- May *significantly* improve performance
- When should you do this:
 - You have a lot of “small” LOB values (values under 8KB) that actually create large rows
 - LOBs aren’t returned on most requests so you’re filling cache with LOB values that aren’t being used

What about “Place Holder” Rows?

Nullability and INSERT Performance

- No default: no specific value required/specified at INSERT
- NULL values DO NOT mean empty space (*NULL bitmap is stored separately from the column data*)
- Working with NULLs
 - Accessing columns which allow NULL values can cause inconsistencies when developers/users are not aware of them
 - Math with NULL values can produce interesting results (value – NULL = NULL)
 - ANSI session settings can affect results sets when accessing columns that allow nulls
- Sometimes it’s best to pre-allocate the row if you’re using place-holders (so that updates do not cause massive fragmentation)

Inconsistencies in Data Types

- **Query doesn't match the column definition**
 - The case of the `implicit_conversion`
- **Key inconsistencies**
 - "Probe Residual" in showplan for hash join
 - May add a hash value for comparisons
 - May add a converted version of a column
 - Wastes storage space, index size, backups, ...
- **Inconsistencies in any layers can be costly**
 - Tables
 - Stored procedures/functions
 - Ad hoc queries/application interface
- **Consider tools like Visual Studio Database Edition for refactoring and static code analysis**

Data Type Best Practices

- **Consistency, consistency, consistency!**
- **Sloppy coding leads to:**
 - Performance problems
 - Difficulty when performance tuning
- **Scalability can only happen with good design**
 - Tables can be created easily but design takes knowledge:
 - Knowing the data
 - Knowing the users
 - Knowing the system
 - Take more time for design/prototyping – the sooner you begin to code, the longer it's going to take!
 - Consider changes over time – if already in place...some tools (Data Dude) might help with refactoring, testing, static code analysis!