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
- 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*
* *If the data type varies:*
  - < 5 chars should be fixed width
  - 5-20 chars – questionable
  - > 20 char – lean towards variable-width
* *For decimal/numeric data:*
  - Find the right range
  - Standardize on decimal or numeric
  - Understand precision and range
  - Consider vardecimal in SQL Server 2005+
* For date/time data
  - Review all choices/ranges in SQL Server 2008+
* *For additional space savings consider:*
  - Compression in SQL Server 2008+
  - Columnstore in SQL Server 2012+
* *Use uniqueidentifier sparingly*
* *Consider "sparse" attribute for 2008+*
  (for Entity Attribute Values [EAV] / flexible design)

Optimal Row Width

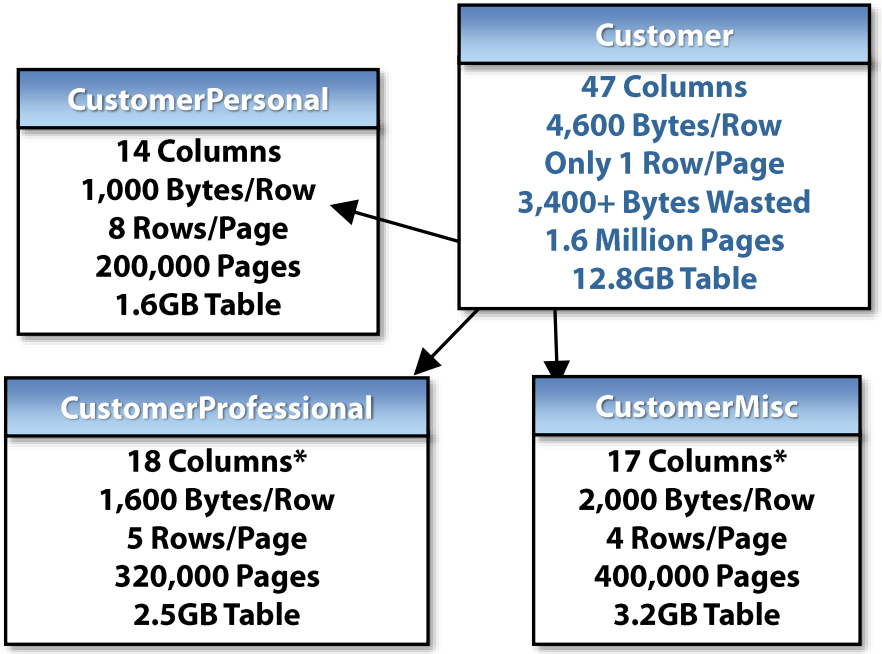
- **Consider table usage above all else**
- **Estimate average row length**
  - Overhead
  - Fixed-width columns
  - Estimate average from realistic sample data

    ```
    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 bytes/page divided by ??? bytes/row = rows/page
- **Calculate wasted bytes – on disk and in memory**

Consider a Customer Table With 1,600,000 Rows



One, singe Customer table = 12.8GB

or

Customer, vertically partitioned into three separate tables = 7.3GB

- Savings in overall disk space (5.5GB saved)
- Not reading data into cache when not necessary
- LOB data can be isolated from more critical data to support online index operations (prior to SQL Server 2012 where rebuilds with LOB can be done online)
- 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 SQL Server 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
- **Set with sp_tableoption**

```
EXEC sp_tableoption tablename
      , 'large value types out of row'
      , TRUE
```

"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)**

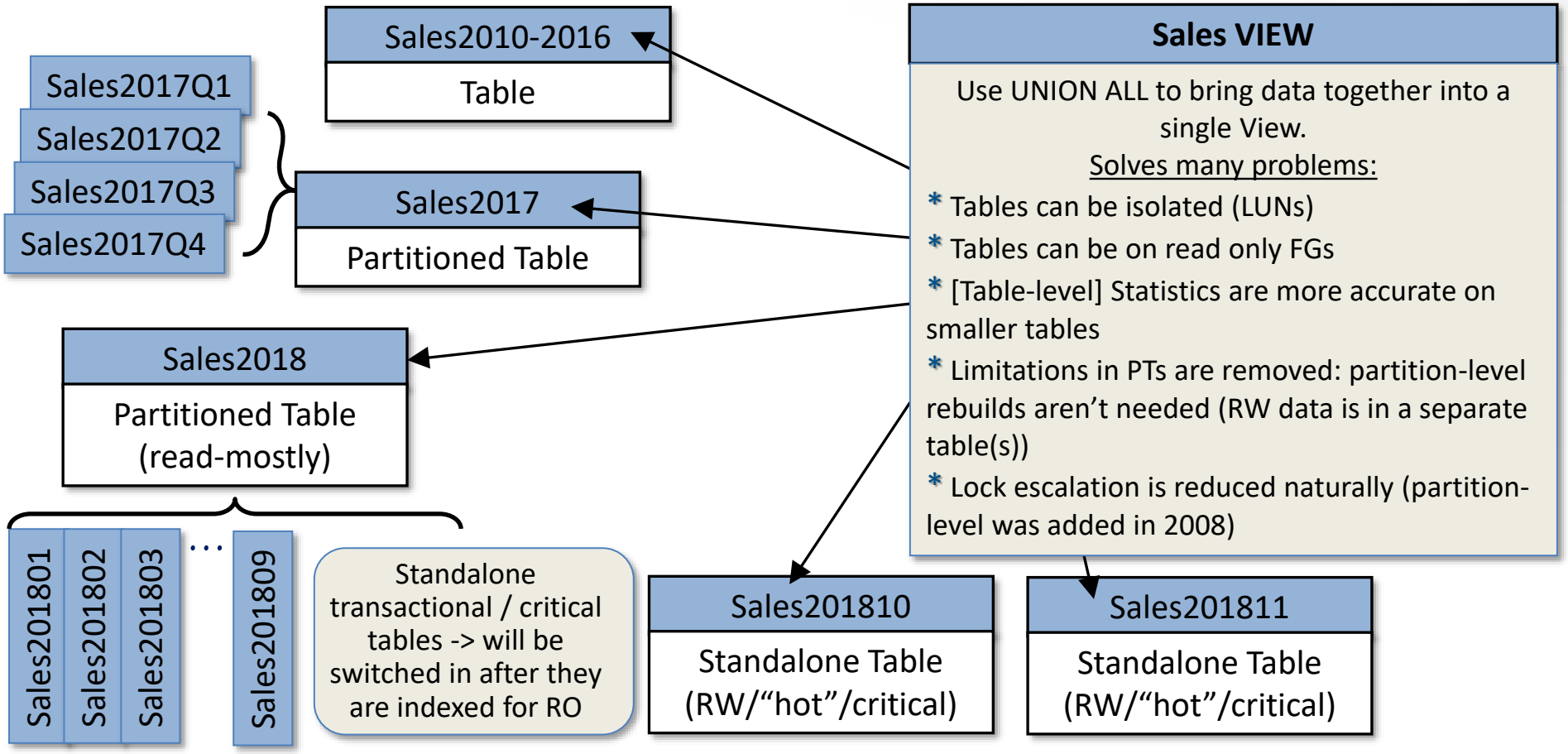
Inconsistencies in Data Types

- **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 for refactoring and static code analysis**

Horizontal / Functionally Partitioning Data

- **Breaking a table into smaller / more manageable chunks to:**
  - Reduce resource contention / limitations
  - Improve options / performance for varying access patterns
  - Allow more maintenance options and reduce costs / restrictions
  - Improve availability and reduce downtime for disaster recovery
  - Remove resource blocking or minimize maintenance costs
- **Usage defines partitioning pattern / partitioning key**
  - Usually date-related (but doesn't have to be)
  - Distinct data patterns in terms of:
    - Usage
    - Criticality
    - Maintenance
- **Queries must specify the partitioning column on every request to aid in partition elimination**

Functionally Partitioning Data



Functionally Partitioning Data

- **Partitioned tables (requirement: Enterprise Edition prior to SQL Server 2016 SP1)**
  - But, for ALL Enterprise ADMIN features such as online operations – you still need EE
  - Can convert an existing table as an ONLINE operation IF the table doesn't have any LOB columns in 2005 / 2008 / R2 (fixed in **2012**)
    - Might run into problems around "unique" index requirements for PTs in that the partitioning column must be a member of the key – for all unique indexes
  - Cannot do fast switching in 2005 if Indexed Views
  - Cannot do fast switching if iFTS desired
- **Partitioned views (benefit: available in any edition)**
  - Might be able to replace an existing table with a view (even for DML) if you meet the correct criteria
    - Might not be able to replace all statements, can programmatically direct modifications (for INSERTs)
  - Conversion may require downtime or time where certain data is inaccessible
  - Definitely more work to architect, manage, design – payoff is often worth it!

Table Design Best Practices

- **Communications, DESIGN, consistency!**
- **Sloppy design (or none!) 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…third-party tools can help with refactoring, testing, static code analysis!