Options for creating indexes

ONLINE = ON	Slower index creation, so don't use this if you don't need to
SORT_IN_TEMPDB = ON	Only makes sense when TempDB is dramatically faster than user database storage, like local SSD or ephemeral SSD
MAXDOP = 0	Can hint an index to be built faster despite server-level settings of MAXDOP 1, like on SharePoint or Dynamics servers
DATA_COMPRESSION = ROW, PAGE, NONE	Only works for on-row data, not MAX, XML, JSON, etc. for columns that end up getting stored off-row. Data warehouse on 2016+? Consider columnstore.

Loading data fast

"Should I drop my indexes before loading?"

Data Loading Performance Guide (2009):
https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008/dd425070(v=sql.100)

Doesn't cover:

- Columnstore indexes
- Database mirroring, AGs
- Azure SQL DB and its variants

Even more guidance:
Google for Paul White minimal logging

Summarizing Minimal Logging Conditions

To assist you in understanding which bulk load operations will be minimally logged and which will not, the following table lists the possible combinations.

Table Indexes	Rows in table	Hints	Without TF 610	With TF 610	Concurrent possible
Heap	Any	TABLOCK	Minimal	Minimal	Yes
Heap	Any	None	Full	Full	Yes
Heap + Index	Any	TABLOCK	Full	Depends (3)	No
Cluster	Empty	TABLOCK, ORDER (1)	Minimal	Minimal	No
Cluster	Empty	None	Full	Minimal	Yes (2)
Cluster	Any	None	Full	Minimal	Yes (2)
Cluster	Any	TABLOCK	Full	Minimal	No
Cluster + Index	Any	None	Full	Depends (3)	Yes (2)
Cluster + Index	Any	TABLOCK	Full	Depends (3)	No

3.5 p6

The fill factor war

Default fill factor: 100% (or 0%, both mean the same thing)

Page splits mean two things:

- Bad split: when SQL Server needs to add/update a row on a page, but there's not enough space left, so it "splits" the page contents into two different pages, or
- Good split: adding a new, empty page for new rows (which isn't a split at all)

The Perfmon counter for page splits includes both kinds

Don't adjust fill factor in order to "fix" page splits unless you're using Extended Events to monitor bad splits on a per-object basis

The fill factor war, continued

When I see...	I say...
Fill factor = 100%	Nothing
Fill factor 80-99%	Nothing, focus on bigger battles
Fill factor 51-79%	"Hey, just FYI, you're wasting X% of your memory because you're leaving X% blank on every page. Plus your backups are taking X% longer, restores, checkdb, index rebuilds, all your maintenance jobs."
Fill factor 1-50%	"I'm going to set fill factor back to a sane number. Wanna start with 80%, 90%, or 100%?"

The fragmentation war

When I see...	I say...
Indexes rebuilt weekly	Nothing
Indexes rebuilt daily, CHECKDB weekly	"How about we switch these schedules and check for corruption daily? We're not keeping our log backups very long, and I'm concerned that we won't catch corruption in time to recover from it."
Indexes rebuilt daily, CHECKDB never done	"We're going to start rotating between index rebuilds and CHECKDB on different days."

3.5 p9

Naming conventions for ISV apps

Independent Software Vendors (ISVs) can label indexes to make it more clear which indexes are theirs, versus the customer's:

- CORE_fieldnames – an index that's a core part of the app's functionality. Everyone likely needs this index.
- JIRA_1234_fieldnames – an index built for a customer for a specific Jira issue, so you can look up why the index is there (and check to see if it's no longer needed)
- FEATURE_fieldnames – indexes required for specific features that not all customers may be using
- ACME_fieldnames – an index custom-build for a specific customer, Acme, for a way they use the product

The presence of indexes can change plans

No nonclustered indexes	More likely to get trivial optimization
The presence of any nonclustered index on a table	Can tip a trivial plan over to full optimization
The presence of any columnstore index on a table	Can trigger batch mode in SQL 2016-17
A computed column using a user-defined function on a table	Causes any query that touches that table to go serial (including CHECKDB), even if the query doesn't touch that column

Signs columnstore will work well

- Tables with 100M+ rows, 100GB+, dozens of columns
- Loaded in batches (like 100k+ rows at a time)
- Not usually updated or deleted
- SQL Server 2016 or newer, 64GB+ RAM
- Data is highly repetitive (compressible)
- Learn more: Columnscore.com,
  my columnstore classes

In-Memory OLTP (Hekaton)

DBCC CHECKDB simply skips these files.

You only find out you have corruption when:

- **Your SQL Server service starts and tries to read all data into RAM (triggering a failed startup, and the entire database is offline)**
- **You read the data pages to back them up (triggering a failed backup, meaning with Hekaton, backup failures are now a critical event)**

Restore and recovery of memory-optimized tables

12/30/2017 • 2 minutes to read • Contributors all

APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse

The basic mechanism to recover or restore a database that uses memory-optimized tables is similar to the mechanism for a database that uses only disk-based tables. But unlike disk-based tables, memory-optimized tables must be loaded into memory before the database is available for user access. This requirement adds a new step in the database recovery.

If the server does not have enough available memory, database recovery fails and the database is marked as suspect. To resolve this problem, see Resolve out-of-memory issues.

Factors that affect load time

During recovery or restore operations, the In-Memory OLTP engine reads data and delta files for loading into physical memory. The load time is determined by:

- The amount of data to load.
- Sequential I/O bandwidth.
- The degree of parallelism, determined by the number of file containers and processor cores.
- The number of log records in the active portion of the log that need to be redone.

3.5 p14

Still interested in Hekaton?

Just go buy the RAM you'd need, and see if that solves the problem.

Estimating memory requirements:
https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/estimate-memory-requirements-for-memory-optimized-tables?view=sql-server-2017

When there is an active workload, additional memory is needed to account for row versioning and various operations. How much memory is needed in practice depends on the workload, but to be safe the recommendation is to start with two times the expected size of memory-optimized tables and indexes, and observe what are the memory requirements in practice. The overhead for row versioning always depends on the characteristics of the workload - especially long-running transactions increase the overhead. For most workloads using larger databases (e.g., >100GB), overhead tends to be limited (25% or less).

3.5 p15