**[Script Produces DML to Fix Index Problems](http://sqlsoundings.blogspot.com/2010/01/script-produces-dml-to-correct-problems.html)**

by [Larry Leonard](http://sqlsoundings.blogspot.com/search/label/SQLServerPedia%20Syndication)

Here's the first version of my *Index\_Analysis.sql* script, which examines all the indexes in the current database and emits DML code to correct any problems it finds. It makes no changes to your database itself - you have to copy and paste the emitted SQL - seemed safer.

This is a work in progress, so the usual caveats apply: do not run this on any database you want to keep, for example, a production database, or any database that isn't backed up. Comments welcome!

[view source](http://sqlserverpedia.com/blog/sql-server-bloggers/script-produces-dml-to-fix-index-problems/#viewSource)



[print](http://sqlserverpedia.com/blog/sql-server-bloggers/script-produces-dml-to-fix-index-problems/#printSource)[?](http://sqlserverpedia.com/blog/sql-server-bloggers/script-produces-dml-to-fix-index-problems/#about)

0001.-------------------------------------------------------------------------------

0002.-- Analyzes the indexes in a database to determine the use, characteristics,

0003.-- and any recommended action for each one (and emits the necessary T-SQL to

0004.-- perform the action). This includes schema modifications, statistics issues

0005.-- and fragmentation issues (including extent fragmentation of heap tables and

0006.-- excessive forwarding pointers).

0007.--

0008.-- Index REORGANIZEs and REBUILDs are always done ONLINE when possible. Only

0009.-- certain SQL Server editions even support ONLINE, and certain kinds of

0010.-- indexes cannot be built ONLINE.

0011.--

0012.-- To populate the emitted T-SQL template code, copy it to a SEPARATE window,

0013.-- and press "Ctrl-Shift-M".

0014.--

0015.-- References: [http://msdn.microsoft.com/en-us/library/ms189858.aspx](http://msdn.microsoft.com/en-us/library/ms189858.aspx%20) ;

0016.-- [http://sqlfool.com/2009/06/index-defrag-script-v30/](http://sqlfool.com/2009/06/index-defrag-script-v30/%20) ;

0017.-- <http://sqlblog.com/blogs/kalen_delaney/archive/2009/11/11/fragmentation-and-forwarded-records-in-a-heap.aspx>

0018.-- [http://msdn.microsoft.com/en-us/library/ms174169.aspx](http://msdn.microsoft.com/en-us/library/ms174169.aspx%20) ;

0019.-- <http://www.sqlskills.com/BLOGS/PAUL/post/Where-do-the-Books-Online-index-fragmentation-thresholds-come-from.aspx>

0020.-- <http://www.codeproject.com/KB/database/OptimizeDBUseIndexing.aspx>

0021.--

0022.-- Revision: 2010-01-13

0023.-------------------------------------------------------------------------------

0024.-- Copyright 2010 Larry Leonard, Definitive Solutions Inc.

0025.-- http://www.LarryLeonard.net ;

0026.--

0027.-- Copying and distribution of this file, with or without modification, are

0028.-- permitted in any medium without royalty provided the copyright notice and

0029.-- this notice are preserved. This file is offered as-is without any warranty.

0030.-------------------------------------------------------------------------------

0031.

0032.

0033.-------------------------------------------------------------------------------

0034.-- To Do List

0035.-------------------------------------------------------------------------------

0036.

0037.--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

0038.-- XML indexes (Primary and secondary)

0039.-- See <http://www.codeproject.com/KB/database/OptimizeDBUseIndexing.aspx>

0040.-- When the primary index on XML column is created, SQL Server shreds the

0041.-- XML content and creates several rows of data that include information

0042.-- like element and attribute names, the path to the root, node types and

0043.-- values, and so on. So, creating the primary index enables SQL server

0044.-- to support XQuery requests more easily.

0045.--

0046.-- Creating the primary XML indexes improves XQuery performance because

0047.-- the XML data is shredded already. But, SQL Server still needs to scan

0048.-- through the shredded data to find the desired result. To further

0049.-- improve query performance, the secondary XML index should be created on

0050.-- top of primary XML indexes.

0051.--

0052.-- Three types of secondary XML indexes:

0053.--

0054.-- - “Path” Secondary XML indexes: Useful when using the .exist()

0055.-- methods to determine whether a specific path exists.

0056.--

0057.-- - “Value” Secondary XML indexes: Used when performing value-based

0058.-- queries where the full path is unknown or includes wildcards.

0059.--

0060.-- - “Property” Secondary XML indexes: Used to retrieve property

0061.-- values when the path to the value is known.

0062.--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

0063.

0064.-- Compression (row and page)

0065.-- Make the temp tables permanent

0066.-- What if table has no UNIQUE index?

0067.-- Check if FK is trusted (sys.foreign\_keys.is\_not\_trusted)

0068.-- Check for indexes that have ghost records (reorg, rebuild, or scan will fix)

0069.-- Try/catch blocks

0070.-- If an index was compressed, must be rebuilt as compressed

0071.-- Reorg does not update column statistics

0072.

0073.-- Handle index creation on a computed column (formula must not contain any

0074.-- nondeterministic functions)

0075.

0076.-- Since forwarding pointers can only occur in heaps, you can get rid of them

0077.-- by creating a clustered index on the table.

0078.

0079.-- Find fulltext indexes to rebuild

0080.-- SELECT OBJECT\_NAME(table\_id) AS TableName,

0081.-- COUNT(fragment\_id) AS Fragments = 30

0082.

0083.-- If you run a query against sysindexes (yes, the backward compatibility view,

0084.-- NOT the new sys.indexes catalog view), then you can see the OrigFillFactor

0085.-- setting for ALL of your indexes.

0086.-------------------------------------------------------------------------------

0087.

0088.

0089.-------------------------------------------------------------------------------

0090.-- MUST be set as shown to support indexes on computed columns, indexed views.

0091.SET ANSI\_NULLS ON -- Deprecated: leave set ON.

0092.SET ANSI\_PADDING ON -- Deprecated: leave set ON.

0093.SET ANSI\_WARNINGS ON -- No trailing blanks saved.

0094.SET ARITHABORT ON -- Math failure not ignored.

0095.SET CONCAT\_NULL\_YIELDS\_NULL ON -- NULL plus string is NULL.

0096.SET NUMERIC\_ROUNDABORT OFF -- Allows loss of precision.

0097.SET QUOTED\_IDENTIFIER ON -- Allows reserved keywords.

0098.

0099.-- These are not, strictly speaking, required, but are generally good practice.

0100.SET NOCOUNT ON -- Minimize network traffic.

0101.SET ROWCOUNT 0 -- Reset in case it got set.

0102.SET XACT\_ABORT ON -- Make transactions behave.

0103.-------------------------------------------------------------------------------

0104.

0105.IF DB\_NAME() IN ('master', 'tempdb', 'model', 'msdb') RAISERROR(' $$$ YOU ARE ATTACHED TO A SYSTEM DB $$$', 20, 1) WITH NOWAIT, LOG

0106.IF 10 > CAST(CAST(SERVERPROPERTY('ProductVersion') AS CHAR(2)) AS INT) RAISERROR(' $$$ REQUIRES SQL SERVER 2008 OR LATER $$$', 20, 1) WITH NOWAIT, LOG

0107.

0108.

0109.-------------------------------------------------------------------------------

0110.-- Variables you can change.

0111.-------------------------------------------------------------------------------

0112.

0113.-- These values are extremely nebulous and unique to each database. Also,

0114.-- regarding the values recommended in BOL, the person who wrote DBCC

0115.-- INDEXDEFRAG and DBCC SHOWCONTIG for SQL Server 2000, Paul S. Randal,

0116.-- says: "I made them up".

0117.

0118.DECLARE @nFillFactor SMALLINT = 80 -- The FILLFACTOR we want to maintain.

0119.DECLARE @nPageFullRebuildValue SMALLINT = 60 -- If less full than this, do a REBUILD...

0120.DECLARE @nPageFullReorgValue SMALLINT = 75 -- ... else if less full than this, do a REORGANIZE.

0121.DECLARE @nExtFragRebuildValue SMALLINT = 15 -- If more externally fragmented than this, do a REBUILD...

0122.DECLARE @nExtFragReorgValue SMALLINT = 10 -- ... else if more externally fragmented than this, do a REORGANIZE.

0123.DECLARE @nSmallestTableToProcessInPages SMALLINT = 1000 -- Small tables won't benefit from REORGANIZE nor REBUILD.

0124.

0125.

0126.-------------------------------------------------------------------------------

0127./\* Debug values.

0128.SET @nFillFactor = 99

0129.SET @nPageFullRebuildValue = 1

0130.SET @nPageFullReorgValue = 5

0131.SET @nExtFragRebuildValue = 5

0132.SET @nExtFragReorgValue = 1

0133.SET @nSmallestTableToProcessInPages = 1

0134. Debug values. \*/

0135.-------------------------------------------------------------------------------

0136.

0137.

0138.-------------------------------------------------------------------------------

0139.-- Drop stale temp tables.

0140.-------------------------------------------------------------------------------

0141.

0142.IF OBJECT\_ID('tempdb..#IndexList') IS NOT NULL DROP TABLE #IndexList

0143.IF OBJECT\_ID('tempdb..#ForeignKeys') IS NOT NULL DROP TABLE #ForeignKeys

0144.

0145.

0146.-------------------------------------------------------------------------------

0147.-- Test input values.

0148.-------------------------------------------------------------------------------

0149.

0150.IF @nSmallestTableToProcessInPages NOT BETWEEN 1 AND 10000

0151. RAISERROR(' $$$ @nSmallestTableToProcessInPages outside recommended range $$$', 20, 1) WITH NOWAIT, LOG

0152.

0153.IF @nPageFullRebuildValue NOT BETWEEN 1 AND 100

0154. RAISERROR(' $$$ @nPageFullRebuildValue outside legal range $$$', 20, 1) WITH NOWAIT, LOG

0155.

0156.IF @nPageFullReorgValue NOT BETWEEN 1 AND 100

0157. RAISERROR(' $$$ @nPageFullReorgValue outside legal range $$$', 20, 1) WITH NOWAIT, LOG

0158.

0159.IF @nFillFactor NOT BETWEEN 1 AND 100

0160. RAISERROR(' $$$ @nFillFactor outside legal range $$$', 20, 1) WITH NOWAIT, LOG

0161.

0162.IF @nExtFragRebuildValue NOT BETWEEN 1 AND 100

0163. RAISERROR(' $$$ @nExtFragRebuildValue outside legal range $$$', 20, 1) WITH NOWAIT, LOG

0164.

0165.IF @nExtFragReorgValue NOT BETWEEN 1 AND 100

0166. RAISERROR(' $$$ @nExtFragReorgValue outside legal range $$$', 20, 1) WITH NOWAIT, LOG

0167.

0168.IF @nPageFullRebuildValue >= @nPageFullReorgValue

0169. RAISERROR(' $$$ @nPageFullRebuildValue must be less than @nPageFullReorgValue $$$', 20, 1) WITH NOWAIT, LOG

0170.

0171.IF @nExtFragReorgValue >= @nExtFragRebuildValue

0172. RAISERROR(' $$$ @nExtFragReorgValue must be less than @nExtFragRebuildValue $$$', 20, 1) WITH NOWAIT, LOG

0173.

0174.IF @nPageFullReorgValue > @nFillFactor

0175. RAISERROR(' $$$ @nPageFullReorgValue must be less than @nFillFactor $$$', 20, 1) WITH NOWAIT, LOG

0176.

0177.

0178.-------------------------------------------------------------------------------

0179.-- Variables you can't change.

0180.-------------------------------------------------------------------------------

0181.

0182.DECLARE @DB\_ID SMALLINT = DB\_ID()

0183.DECLARE @DB\_Name sysname = DB\_NAME(@DB\_ID)

0184.DECLARE @PAGES\_PER\_EXTENT SMALLINT = 8

0185.DECLARE @sFillFactor NVARCHAR(8) = CAST(@nFillFactor AS NVARCHAR(8))

0186.

0187.DECLARE @sOnlineRebuild NVARCHAR(8) =

0188. CASE SERVERPROPERTY('Edition')

0189. WHEN 'Desktop Engine' THEN 'OFF'

0190. WHEN 'Developer Edition' THEN 'ON'

0191. WHEN 'Enterprise Edition' THEN 'ON'

0192. WHEN 'Enterprise Evaluation Edition' THEN 'ON'

0193. WHEN 'Personal Edition' THEN 'OFF'

0194. WHEN 'Standard Edition' THEN 'OFF'

0195. ELSE 'OFF'

0196. END

0197.

0198.RAISERROR('Processing database %d (%s)', 10, 1, @DB\_ID, @DB\_Name) WITH NOWAIT

0199.

0200.

0201.-------------------------------------------------------------------------------

0202.-- Add all FK's to their temp table.

0203.-------------------------------------------------------------------------------

0204.

0205.RAISERROR('Finding all foreign keys', 10, 1) WITH NOWAIT

0206.

0207.CREATE TABLE #ForeignKeys

0208.(

0209. fk\_name sysname NOT NULL DEFAULT ('')

0210. , table\_object\_id INT NOT NULL DEFAULT (0)

0211. , fk\_columns NVARCHAR(MAX) NOT NULL DEFAULT ('')

0212. , fk\_columns\_compare NVARCHAR(MAX) NOT NULL DEFAULT ('')

0213.)

0214.

0215.INSERT INTO #ForeignKeys

0216.(

0217. fk\_name

0218. , table\_object\_id

0219. , fk\_columns

0220. , fk\_columns\_compare

0221.)

0222. SELECT

0223. fk.name + ' | REFERENCING' AS fk\_name

0224. , fkc.parent\_object\_id AS table\_object\_id

0225. , STUFF((SELECT ', ' + QUOTENAME(c.name)

0226. FROM sys.foreign\_key\_columns ifkc WITH (NOLOCK)

0227. JOIN sys.columns c WITH (NOLOCK)

0228. ON ifkc.parent\_object\_id = c.object\_id

0229. AND ifkc.parent\_column\_id = c.column\_id

0230. WHERE fk.object\_id = ifkc.constraint\_object\_id

0231. ORDER BY ifkc.constraint\_column\_id

0232. FOR XML PATH('')), 1, 2, '') AS fk\_columns

0233. , (SELECT QUOTENAME(ifkc.parent\_column\_id,'(')

0234. FROM sys.foreign\_key\_columns ifkc WITH (NOLOCK)

0235. WHERE fk.object\_id = ifkc.constraint\_object\_id

0236. ORDER BY ifkc.constraint\_column\_id

0237. FOR XML PATH('')) AS fk\_columns\_compare

0238. FROM sys.foreign\_keys fk WITH (NOLOCK)

0239. JOIN sys.foreign\_key\_columns fkc WITH (NOLOCK)

0240. ON fk.object\_id = fkc.constraint\_object\_id

0241. WHERE fkc.constraint\_column\_id = 1

0242.

0243.UNION ALL

0244.

0245.SELECT

0246. fk.name + ' | REFERENCED' AS fk\_name

0247. , fkc.referenced\_object\_id AS object\_id

0248. , STUFF((SELECT ', ' + QUOTENAME(c.name)

0249. FROM sys.foreign\_key\_columns ifkc WITH (NOLOCK)

0250. JOIN sys.columns c WITH (NOLOCK)

0251. ON ifkc.referenced\_object\_id = c.object\_id

0252. AND ifkc.referenced\_column\_id = c.column\_id

0253. WHERE fk.object\_id = ifkc.constraint\_object\_id

0254. ORDER BY ifkc.constraint\_column\_id

0255. FOR XML PATH('')), 1, 2, '') AS fk\_columns

0256. , (SELECT QUOTENAME(ifkc.referenced\_column\_id, '(')

0257. FROM sys.foreign\_key\_columns ifkc WITH (NOLOCK)

0258. WHERE fk.object\_id = ifkc.constraint\_object\_id

0259. ORDER BY ifkc.constraint\_column\_id

0260. FOR XML PATH('')) AS fk\_columns\_compare

0261. FROM sys.foreign\_keys fk WITH (NOLOCK)

0262. JOIN sys.foreign\_key\_columns fkc WITH (NOLOCK)

0263. ON fk.object\_id = fkc.constraint\_object\_id

0264. WHERE fkc.constraint\_column\_id = 1

0265.

0266.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0267.

0268.

0269.-------------------------------------------------------------------------------

0270.-- Create and populate the index table.

0271.-------------------------------------------------------------------------------

0272.

0273.RAISERROR('Finding all indexes', 10, 1) WITH NOWAIT

0274.

0275.CREATE TABLE #IndexList

0276.(

0277. -- IDENTIFICATION

0278. id INT IDENTITY(1,1) NOT NULL

0279. , database\_id INT NULL DEFAULT(-1)

0280. , database\_name sysname NULL DEFAULT('')

0281. , schema\_id INT NULL DEFAULT(-1)

0282. , schema\_name sysname NULL DEFAULT('')

0283. , table\_object\_id INT NULL DEFAULT(-1)

0284. , table\_name sysname NULL DEFAULT('')

0285.

0286. -- INDEX INFO

0287. , index\_id INT NULL DEFAULT(-1)

0288. , index\_name NVARCHAR(256) NULL DEFAULT('')

0289. , partition\_number SMALLINT NULL DEFAULT(-1)

0290. , index\_type\_desc NVARCHAR(67) NULL DEFAULT('')

0291. , is\_unique BIT NULL DEFAULT(0)

0292.

0293. -- TABLE INFO

0294. , table\_has\_unique BIT NULL DEFAULT(0)

0295. , table\_has\_lob BIT NULL DEFAULT(0)

0296. , table\_has\_clustered BIT NULL DEFAULT(0)

0297. , table\_is\_partitioned BIT NULL DEFAULT(0)

0298.

0299. -- MEMORY

0300.

0301. -- Total number of pages reserved for the index.

0302. , reserved\_page\_count BIGINT NULL DEFAULT(0)

0303.

0304. -- The amount of space in MB the index utilizes on disk.

0305. , size\_in\_mb DECIMAL(12, 2) NULL DEFAULT(0)

0306.

0307. -- Total number of pages in the buffer for the index.

0308. , buffered\_page\_count INT NULL DEFAULT(0)

0309.

0310. -- The amount of space in MBs in the buffer for the index.

0311. , buffer\_mb DECIMAL(12, 2) NULL DEFAULT(0)

0312.

0313. -- The percentage of an index that is current in the SQL Server buffer.

0314. , pct\_in\_buffer DECIMAL(12, 2) NULL DEFAULT(0)

0315.

0316. -- The amount of space in MB in the SQL Server buffer that is

0317. -- being utilized by the table.

0318. , table\_buffer\_mb DECIMAL(12, 2) NULL DEFAULT(0)

0319.

0320. -- USAGE

0321.

0322. -- Total number of seek, scan, and lookup operations for the index.

0323. , user\_total BIGINT NULL DEFAULT(0)

0324.

0325. -- Percentage of total number of seek, scan, and lookup operations

0326. -- for this index compared to all seek, scan, and lookup operations

0327. -- for existing indexes for the table.

0328. , user\_total\_pct DECIMAL(6, 2) NULL DEFAULT(0)

0329.

0330. -- Percentage of total number of seek, scan, and lookup operations

0331. -- for this index compared to all seek, scan, and lookup operations

0332. -- for existing and potential indexes for the table. This number

0333. -- is naturally skewed because a seek for potential Index A

0334. -- resulted in another operation on an existing index and both of

0335. -- these operations would be counted.

0336. , estimated\_user\_total\_pct DECIMAL(6, 2) NULL DEFAULT(0)

0337.

0338. -- A single row retreival.

0339. , user\_seeks BIGINT NULL DEFAULT(0)

0340.

0341. -- A range of values, or a non-unique query.

0342. , user\_scans BIGINT NULL DEFAULT(0)

0343.

0344. -- Clustered indexes only - bookmark lookups.

0345. , user\_lookups BIGINT NULL DEFAULT(0)

0346.

0347. -- Writes to the index caused by inserts, updates, or deletes.

0348. , user\_updates BIGINT NULL DEFAULT(0)

0349.

0350. -- LOCKING

0351.

0352. -- Cumulative number of row locks requested.

0353. , row\_lock\_count BIGINT NULL DEFAULT(0)

0354.

0355. -- Cumulative number of times the Database Engine waited on a row lock.

0356. , row\_lock\_wait\_count BIGINT NULL DEFAULT(0)

0357.

0358. -- Total number of milliseconds the Database Engine waited on a row lock.

0359. , row\_lock\_wait\_in\_ms BIGINT NULL DEFAULT(0)

0360.

0361. -- Percentage of row locks that encounter waits on a row lock.

0362. , row\_block\_pct DECIMAL(6, 2) NULL DEFAULT(0)

0363.

0364. -- Average number of milliseconds the Database Engine waited on a row lock.

0365. , avg\_row\_lock\_waits\_ms BIGINT NULL DEFAULT(0)

0366.

0367. -- COLUMNS INFO

0368.

0369. -- Columns that are part of the index, missing index or foreign key.

0370. , indexed\_columns NVARCHAR(MAX) NULL DEFAULT('')

0371.

0372. -- Columns that are included in the index or missing index.

0373. , included\_columns NVARCHAR(MAX) NULL DEFAULT('')

0374.

0375. -- Column IDs that are part of the index, missing index or foreign key.

0376. , indexed\_columns\_compare NVARCHAR(MAX) NULL DEFAULT('')

0377.

0378. -- Column IDs that are included in the index or missing index.

0379. , included\_columns\_compare NVARCHAR(MAX) NULL DEFAULT('')

0380.

0381. -- PROBLEMS

0382.

0383. -- List of indexes that exist on table that are identical to index on this row.

0384. , duplicate\_indexes NVARCHAR(MAX) NULL DEFAULT('')

0385.

0386. -- List of indexes that exist on the table that overlap the index on this row.

0387. , overlapping\_indexes NVARCHAR(MAX) NULL DEFAULT('')

0388.

0389. -- FOREIGN KEYS

0390.

0391. -- List of foreign keys that are related to the index either as an

0392. -- exact match or covering index.

0393. , related\_foreign\_keys NVARCHAR(MAX) NULL DEFAULT('')

0394.

0395. -- XML document listing foreign keys that are related to the index

0396. -- either as an exact match or covering index.

0397. , related\_foreign\_keys\_xml XML NULL

0398.

0399. -- STATISTICS

0400. , date\_stats\_updated SMALLDATETIME NULL DEFAULT(0)

0401.

0402. -- Number of rows in the index.

0403. , row\_count BIGINT NULL DEFAULT(0)

0404. , row\_mod\_count BIGINT NULL DEFAULT(0)

0405.

0406. -- Number of rows in the index that must be modified before statistics are rebuilt.

0407. , row\_stat\_threshold BIGINT NULL DEFAULT(-1)

0408.

0409. -- FRAGMENTATION

0410.

0411. -- The number of 8 KB pages used on disk.

0412. , disk\_page\_count BIGINT NULL DEFAULT(0)

0413.

0414. -- "Logical/external fragmentation" for indexes. For heaps, this

0415. -- value is actually the percentage of "extent fragmentation".

0416. , avg\_fragmentation\_in\_percent BIGINT NULL DEFAULT(0)

0417.

0418. -- "Internal fragmentation" is when the index's pages are not

0419. -- filled to the current fill factor level. To achieve optimal

0420. -- disk space use, this value should be close to 100 percent for

0421. -- an index that will not have many random inserts. However, an

0422. -- index that has many random inserts and has very full pages will

0423. -- have an increased number of page splits. This causes more

0424. -- fragmentation. Therefore, in order to reduce page splits, the

0425. -- value should be less than 100 percent.

0426. , avg\_page\_space\_used\_in\_percent BIGINT NULL DEFAULT(0)

0427.

0428. -- RECOMMENDATIONS

0429.

0430. -- Calculation of effect of a potential index, based on the seeks

0431. -- and scans that the index could have supported multiplied by

0432. -- average improvement the index would have provided. This is

0433. -- included only for missing indexes.

0434. , impact INT NULL DEFAULT(-1)

0435.

0436. -- Ranking of the existing indexes ordered by user\_total

0437. -- descending across the indexes for the table.

0438. , rank BIGINT NULL DEFAULT(-1)

0439. , index\_action NVARCHAR(32) NULL DEFAULT('')

0440. , dml\_for\_action NVARCHAR(MAX) NULL DEFAULT('')

0441.)

0442.

0443.-- Now that the index table is created, add rows.

0444.; WITH

0445.cteAllocationUnits

0446.AS

0447.(

0448. -- One row for each allocation unit in the database for standard

0449. -- non-LOB data. (HoBT = heap or b-tree)

0450. SELECT p.object\_id AS table\_object\_id

0451. , p.index\_id AS index\_id

0452. , p.partition\_number AS partition\_number

0453. , au.allocation\_unit\_id AS allocation\_unit\_id

0454. FROM sys.allocation\_units au WITH (NOLOCK)

0455. JOIN sys.partitions p WITH (NOLOCK)

0456. ON au.container\_id = p.hobt\_id

0457. AND (au.type\_desc = 'IN\_ROW\_DATA' OR au.type\_desc = 'ROW\_OVERFLOW\_DATA')

0458. WHERE object\_id > 100

0459.

0460. UNION ALL

0461.

0462. -- One row for each allocation unit in the database for LOB data.

0463. SELECT p.object\_id AS table\_object\_id

0464. , p.index\_id AS index\_id

0465. , p.partition\_number AS partition\_number

0466. , au.allocation\_unit\_id AS allocation\_unit\_id

0467. FROM sys.allocation\_units au WITH (NOLOCK)

0468. JOIN sys.partitions p WITH (NOLOCK)

0469. ON au.container\_id = p.partition\_id

0470. AND au.type\_desc = 'LOB\_DATA'

0471. WHERE object\_id > 100

0472.)

0473.,

0474.cteMemoryBuffer

0475.AS

0476.(

0477. -- Get data on allocation units for each index on all of its partitions

0478. -- re buffer usage. One row for each index-partition.

0479. SELECT au.table\_object\_id AS object\_id

0480. , au.index\_id AS index\_id

0481. , au.partition\_number AS partition\_number

0482. , COUNT(\*) AS buffered\_page\_count

0483. , CONVERT(DECIMAL(12,2), CAST(COUNT(\*) AS BIGINT) \* CAST(8 AS FLOAT) / 1024) AS buffer\_mb

0484. FROM sys.dm\_os\_buffer\_descriptors bd WITH (NOLOCK)

0485. JOIN cteAllocationUnits au WITH (NOLOCK)

0486. ON bd.allocation\_unit\_id = au.allocation\_unit\_id

0487. WHERE bd.database\_id = @DB\_ID

0488. GROUP BY au.table\_object\_id,

0489. au.index\_id,

0490. au.partition\_number

0491.)

0492.,

0493.cteFragInfo

0494.AS

0495.(

0496. -- We look at all node levels, not just the leaf. We include heap tables

0497. -- to get their extent fragmentation.

0498. SELECT ips.object\_id AS table\_object\_id

0499. , ips.index\_id AS index\_id

0500. , ips.page\_count AS disk\_page\_count

0501. , ips.avg\_fragmentation\_in\_percent AS avg\_fragmentation\_in\_percent -- External fragmentation (or, extent fragmentation for heaps).

0502. , ips.avg\_page\_space\_used\_in\_percent AS avg\_page\_space\_used\_in\_percent -- Internal fragmentation.

0503.

0504. -- For an index, one row for each level of the B-tree in each partition.

0505. -- For a heap, one row for the IN\_ROW\_DATA allocation unit of each partition.

0506. -- For LOB data, one row for the LOB\_DATA allocation unit of each partition.

0507. FROM sys.dm\_db\_index\_physical\_stats

0508. (

0509. @DB\_ID, -- For this database only.

0510. NULL, -- For all tables and views in the specified database.

0511. NULL, -- For all indexes for a base table or view.

0512. NULL, -- For all partitions of the owning object (index, heap, or LOB).

0513. 'DETAILED' -- Scans all pages and returns all statistics.

0514. ) ips

0515.

0516. LEFT OUTER JOIN sys.dm\_db\_index\_usage\_stats ius -- Rows with no usage since the

0517. ON ius.database\_id = ips.database\_id -- last restart will be null.

0518. AND ius.object\_id = ips.object\_id

0519. AND ius.index\_id = ips.index\_id

0520. WHERE ips.object\_id > 100 -- Ignore system tables.

0521. AND ips.page\_count > @PAGES\_PER\_EXTENT -- Ignore small tables as specified by user.

0522. AND ips.page\_count > 1 -- Regardless of value set for @PAGES\_PER\_EXTENT,

0523. -- always ignore tables that live on a mixed extent.

0524.)

0525.,

0526.cteStatsInfo

0527.AS

0528.(

0529. -- Stats info for every index on every table.

0530. SELECT i.object\_id AS table\_object\_id

0531. , i.index\_id AS index\_id

0532. , MIN(STATS\_DATE(s.object\_id, s.stats\_id)) AS date\_stats\_updated

0533. , MIN(si.rowmodctr) AS row\_mod\_count

0534. , CASE

0535. WHEN MAX(i.index\_id) = 0 THEN -1 -- Heaps don't have stats.

0536. WHEN MIN(si.rowcnt) > 500 THEN 500 + (MIN(si.rowcnt) / 5)

0537. ELSE 500

0538. END AS row\_stat\_threshold

0539. FROM sys.indexes i WITH (NOLOCK)

0540. JOIN sys.index\_columns ic WITH (NOLOCK)

0541. ON i.object\_id = ic.object\_id

0542. AND i.index\_id = ic.index\_id

0543. JOIN sys.sysindexes si WITH (NOLOCK)

0544. ON i.object\_id = si.id

0545. AND i.index\_id = si.indid

0546.left outer JOIN sys.stats\_columns sc WITH (NOLOCK)

0547. ON ic.object\_id = sc.object\_id

0548. AND ic.column\_id = sc.column\_id

0549.left outer JOIN sys.stats s WITH (NOLOCK)

0550. ON sc.object\_id = s.object\_id

0551. AND sc.stats\_id = s.stats\_id

0552. WHERE i.object\_id > 100

0553. AND ISNULL(i.name, '') ''

0554. GROUP BY i.object\_id,

0555. i.index\_id

0556.)

0557.

0558.INSERT INTO #IndexList

0559.(

0560. database\_id

0561. , database\_name

0562. , schema\_id

0563. , schema\_name

0564. , table\_object\_id

0565. , table\_name

0566. , index\_id

0567. , index\_name

0568. , is\_unique

0569. , index\_type\_desc

0570. , partition\_number

0571. , reserved\_page\_count

0572. , size\_in\_mb

0573. , buffered\_page\_count

0574. , buffer\_mb

0575. , pct\_in\_buffer

0576. , row\_count

0577. , rank

0578. , user\_total

0579. , user\_total\_pct

0580. , user\_seeks

0581. , user\_scans

0582. , user\_lookups

0583. , user\_updates

0584. , row\_lock\_count

0585. , row\_lock\_wait\_count

0586. , row\_lock\_wait\_in\_ms

0587. , row\_block\_pct

0588. , avg\_row\_lock\_waits\_ms

0589. , indexed\_columns

0590. , included\_columns

0591. , indexed\_columns\_compare

0592. , included\_columns\_compare

0593. , disk\_page\_count

0594. , avg\_fragmentation\_in\_percent

0595. , avg\_page\_space\_used\_in\_percent

0596. , date\_stats\_updated

0597. , row\_mod\_count

0598. , row\_stat\_threshold

0599.)

0600.SELECT

0601. @DB\_ID AS database\_id

0602. , @DB\_Name AS database\_name

0603. , s.schema\_id AS schema\_id

0604. , s.name AS schema\_name

0605. , t.object\_id AS table\_object\_id

0606. , t.name AS table\_name

0607. , COALESCE(i.index\_id, -1) AS index\_id

0608. , COALESCE(i.name, '') AS index\_name

0609. , COALESCE(i.is\_unique, 0) AS is\_unique

0610. , CASE WHEN i.is\_unique = 1 THEN 'UNIQUE ' ELSE '' END + i.type\_desc AS index\_type\_desc

0611. , ps.partition\_number AS partition\_number

0612. , ps.reserved\_page\_count AS reserved\_page\_count

0613. , CAST(reserved\_page\_count \* CAST(8 AS FLOAT) / 1024 AS DECIMAL(12,2)) AS size\_in\_mb

0614. , COALESCE(mb.buffered\_page\_count, 0) AS buffered\_page\_count

0615. , COALESCE(mb.buffer\_mb, 0) AS buffer\_mb

0616. , COALESCE(CAST(100 \* buffer\_mb /

0617. NULLIF(CAST(reserved\_page\_count \*

0618. CAST(8 AS FLOAT) / 1024 AS DECIMAL(12,2)), 0) AS DECIMAL(12,2)), 0) AS pct\_in\_buffer

0619. , row\_count AS row\_count

0620. , ROW\_NUMBER() OVER (PARTITION BY i.object\_id

0621. ORDER BY i.is\_primary\_key DESC,

0622. ius.user\_seeks + ius.user\_scans + ius.user\_lookups DESC) AS rank

0623. , COALESCE(ius.user\_seeks + ius.user\_scans + ius.user\_lookups, 0) AS user\_total

0624. , COALESCE(CAST(100 \* (ius.user\_seeks + ius.user\_scans + ius.user\_lookups) /

0625. (NULLIF(SUM (ius.user\_seeks + ius.user\_scans + ius.user\_lookups)

0626. OVER (PARTITION BY i.object\_id), 0) \* 1.0) AS DECIMAL(6,2)), 0) AS user\_total\_pct

0627. , COALESCE(ius.user\_seeks, 0) AS user\_seeks

0628. , COALESCE(ius.user\_scans, 0) AS user\_scans

0629. , COALESCE(ius.user\_lookups, 0) AS user\_lookups

0630. , COALESCE(ius.user\_updates, 0) AS user\_updates

0631. , COALESCE(ios.row\_lock\_count, 0) AS row\_lock\_count

0632. , COALESCE(ios.row\_lock\_wait\_count, 0) AS row\_lock\_wait\_count

0633. , COALESCE(ios.row\_lock\_wait\_in\_ms, 0) AS row\_lock\_wait\_in\_ms

0634. , COALESCE(CAST(100.0 \* ios.row\_lock\_wait\_count /

0635. NULLIF(ios.row\_lock\_count, 0) AS DECIMAL(12,2)), 0) AS row\_block\_pct

0636. , COALESCE(CAST(1.0 \* ios.row\_lock\_wait\_in\_ms /

0637. NULLIF(ios.row\_lock\_wait\_count, 0) AS DECIMAL(12,2)), 0) AS avg\_row\_lock\_waits\_ms

0638.

0639. , COALESCE(STUFF((SELECT ', ' + QUOTENAME(c.name)

0640. FROM sys.index\_columns ic WITH (NOLOCK)

0641. JOIN sys.columns c WITH (NOLOCK)

0642. ON ic.object\_id = c.object\_id

0643. AND ic.column\_id = c.column\_id

0644. WHERE i.object\_id = ic.object\_id

0645. AND i.index\_id = ic.index\_id

0646. AND is\_included\_column = 0

0647. ORDER BY key\_ordinal

0648. FOR XML PATH('')), 1, 2, ''), '') AS indexed\_columns

0649.

0650. , COALESCE(STUFF((SELECT ', ' + QUOTENAME(c.name)

0651. FROM sys.index\_columns ic WITH (NOLOCK)

0652. JOIN sys.columns c WITH (NOLOCK)

0653. ON ic.object\_id = c.object\_id

0654. AND ic.column\_id = c.column\_id

0655. WHERE i.object\_id = ic.object\_id

0656. AND i.index\_id = ic.index\_id

0657. AND is\_included\_column = 1

0658. ORDER BY key\_ordinal

0659. FOR XML PATH('')), 1, 2, ''), '') AS included\_columns

0660.

0661. , COALESCE((SELECT QUOTENAME(ic.column\_id, '(')

0662. FROM sys.index\_columns ic WITH (NOLOCK)

0663. WHERE i.object\_id = ic.object\_id

0664. AND i.index\_id = ic.index\_id

0665. AND is\_included\_column = 0

0666. ORDER BY key\_ordinal

0667. FOR XML PATH('')), '') AS indexed\_columns\_compare

0668.

0669. , COALESCE((SELECT QUOTENAME(ic.column\_id, '(')

0670. FROM sys.index\_columns ic WITH (NOLOCK)

0671. WHERE i.object\_id = ic.object\_id

0672. AND i.index\_id = ic.index\_id

0673. AND is\_included\_column = 1

0674. ORDER BY key\_ordinal

0675. FOR XML PATH('')), SPACE(0)) AS included\_columns\_compare

0676.

0677. , COALESCE(fi.disk\_page\_count, 0) AS page\_count

0678. , COALESCE(fi.avg\_fragmentation\_in\_percent, 0) AS avg\_fragmentation\_in\_percent

0679. , COALESCE(fi.avg\_page\_space\_used\_in\_percent, 0) AS avg\_page\_space\_used\_in\_percent

0680. , COALESCE(si.date\_stats\_updated, 0) AS date\_stats\_updated

0681. , COALESCE(si.row\_mod\_count, 0) AS row\_mod\_count

0682. , COALESCE(si.row\_stat\_threshold, 0) AS row\_stat\_threshold

0683. FROM sys.tables t WITH (NOLOCK)

0684. JOIN sys.schemas s

0685. ON t.schema\_id = s.schema\_id

0686. JOIN sys.indexes i

0687. ON t.object\_id = i.object\_id

0688. JOIN sys.dm\_db\_partition\_stats ps

0689. ON i.object\_id = ps.object\_id

0690. AND i.index\_id = ps.index\_id

0691. LEFT OUTER JOIN sys.dm\_db\_index\_usage\_stats ius

0692. ON i.object\_id = ius.object\_id

0693. AND i.index\_id = ius.index\_id

0694. AND ius.database\_id = @DB\_ID

0695. LEFT OUTER JOIN sys.dm\_db\_index\_operational\_stats

0696. (@DB\_ID, NULL, NULL, NULL) ios

0697. ON ps.object\_id = ios.object\_id

0698. AND ps.index\_id = ios.index\_id

0699. AND ps.partition\_number = ios.partition\_number

0700. LEFT OUTER JOIN cteMemoryBuffer mb

0701. ON ps.object\_id = mb.object\_id

0702. AND ps.index\_id = mb.index\_id

0703. AND ps.partition\_number = mb.partition\_number

0704. LEFT OUTER JOIN cteFragInfo fi

0705. ON ps.object\_id = fi.table\_object\_id

0706. AND ps.index\_id = fi.index\_id

0707. LEFT OUTER JOIN cteStatsInfo si

0708. ON ps.object\_id = si.table\_object\_id

0709. AND ps.index\_id = si.index\_id

0710. WHERE t.is\_ms\_shipped = 0

0711.

0712.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0713.

0714.

0715.-------------------------------------------------------------------------------

0716.-- Insert missing indexes based on user access patterns.

0717.-------------------------------------------------------------------------------

0718.

0719.RAISERROR('Finding missing indexes based on user access patterns', 10, 1) WITH NOWAIT

0720.

0721.INSERT INTO #IndexList

0722.(

0723. database\_id

0724. , database\_name

0725. , schema\_id

0726. , schema\_name

0727. , index\_id

0728. , table\_object\_id

0729. , table\_name

0730. , index\_name

0731. , index\_type\_desc

0732. , impact

0733. , user\_total

0734. , user\_seeks

0735. , user\_scans

0736. , user\_lookups

0737. , indexed\_columns

0738. , included\_columns

0739.)

0740.SELECT

0741. @DB\_ID AS database\_id

0742. , @DB\_Name AS database\_name

0743. , s.schema\_id AS schema\_id

0744. , s.name AS schema\_name

0745. , t.object\_id AS table\_object\_id

0746. , t.name AS table\_name

0747. , -1 AS index\_id

0748. , 'IX\_' + t.name + '\_' + COALESCE(equality\_columns, '') AS index\_name

0749. , 'MISSING (USAGE)' AS index\_type\_desc

0750. , COALESCE((migs.user\_seeks + migs.user\_scans) \* migs.avg\_user\_impact, 0) AS impact

0751. , COALESCE(migs.user\_seeks + migs.user\_scans, 0) AS user\_total

0752. , COALESCE(migs.user\_seeks, 0) AS user\_scans

0753. , 0 AS user\_lookups

0754. , COALESCE(migs.user\_scans, 0) AS user\_seeks

0755. , COALESCE(equality\_columns + ', ', SPACE(0)) +

0756. COALESCE(inequality\_columns, SPACE(0)) AS indexed\_columns

0757. , COALESCE(included\_columns, SPACE(0)) AS included\_columns

0758. FROM sys.tables t WITH (NOLOCK)

0759. JOIN sys.schemas s WITH (NOLOCK)

0760. ON t.schema\_id = s.schema\_id

0761. JOIN sys.dm\_db\_missing\_index\_details mid WITH (NOLOCK)

0762. ON t.object\_id = mid.object\_id

0763. JOIN sys.dm\_db\_missing\_index\_groups mig WITH (NOLOCK)

0764. ON mid.index\_handle = mig.index\_handle

0765. JOIN sys.dm\_db\_missing\_index\_group\_stats migs WITH (NOLOCK)

0766. ON mig.index\_group\_handle = migs.group\_handle

0767. WHERE mid.database\_id = @DB\_ID

0768. AND t.object\_id > 100

0769.

0770.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0771.

0772.

0773.-------------------------------------------------------------------------------

0774.-- Self-updating for duplicate and overlapping FK indexes.

0775.-------------------------------------------------------------------------------

0776.

0777.RAISERROR('Finding duplicate and overlapping FK indexes', 10, 1) WITH NOWAIT

0778.

0779.UPDATE il SET

0780. duplicate\_indexes = COALESCE(STUFF((

0781. SELECT ', ' + index\_name AS [data()]

0782. FROM #IndexList iibl WITH (NOLOCK)

0783. WHERE il.table\_object\_id = iibl.table\_object\_id

0784. AND il.index\_id != iibl.index\_id

0785. AND il.indexed\_columns\_compare = iibl.indexed\_columns\_compare

0786. AND il.included\_columns\_compare = iibl.included\_columns\_compare

0787. FOR XML PATH('')), 1, 2, ''), ''),

0788.

0789. overlapping\_indexes = COALESCE(STUFF((

0790. SELECT ', ' + index\_name AS [data()]

0791. FROM #IndexList iibl WITH (NOLOCK)

0792. WHERE il.table\_object\_id = iibl.table\_object\_id

0793. AND il.index\_id != iibl.index\_id

0794. AND (il.indexed\_columns\_compare LIKE iibl.indexed\_columns\_compare + '%'

0795. OR iibl.indexed\_columns\_compare LIKE il.indexed\_columns\_compare + '%')

0796. AND il.indexed\_columns\_compare != iibl.indexed\_columns\_compare

0797. FOR XML PATH('')), 1, 2, ''), ''),

0798.

0799. related\_foreign\_keys = COALESCE(STUFF((

0800. SELECT ', ' + fk\_name AS [data()]

0801. FROM #ForeignKeys fk WITH (NOLOCK)

0802. WHERE fk.table\_object\_id = il.table\_object\_id

0803. AND il.indexed\_columns\_compare LIKE fk.fk\_columns\_compare + '%'

0804. FOR XML PATH('')), 1, 2, ''), ''),

0805.

0806. related\_foreign\_keys\_xml = COALESCE(CAST((

0807. SELECT fk\_name AS [data()]

0808. FROM #ForeignKeys ifk WITH (NOLOCK)

0809. WHERE ifk.table\_object\_id = il.table\_object\_id

0810. AND il.indexed\_columns\_compare LIKE ifk.fk\_columns\_compare + '%'

0811. FOR XML AUTO) AS XML), '')

0812.

0813.FROM #IndexList il

0814.

0815.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0816.

0817.

0818.-------------------------------------------------------------------------------

0819.-- Self-update #IndexList using calculations.

0820.-------------------------------------------------------------------------------

0821.

0822.RAISERROR('Calculating summary index usage data', 10, 1) WITH NOWAIT

0823.

0824.; WITH cteAggregation

0825.AS (

0826. SELECT

0827. id AS id

0828. , CAST(100.0 \* (user\_seeks + user\_scans + user\_lookups) /

0829. (NULLIF (SUM (user\_seeks + user\_scans + user\_lookups)

0830. OVER(PARTITION BY schema\_name, table\_name), 0) \* 1.0) AS DECIMAL(12,2)) AS estimated\_user\_total\_pct

0831. , SUM(buffer\_mb) OVER (PARTITION BY schema\_name, table\_name) AS table\_buffer\_mb

0832. FROM #IndexList WITH (NOLOCK)

0833.)

0834.UPDATE il

0835. SET estimated\_user\_total\_pct = COALESCE(a.estimated\_user\_total\_pct, 0),

0836. table\_buffer\_mb = a.table\_buffer\_mb

0837. FROM #IndexList il

0838. JOIN cteAggregation a

0839. ON il.id = a.id

0840.

0841.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0842.

0843.

0844.-------------------------------------------------------------------------------

0845.-- Insert any missing FK indexes to #IndexList.

0846.-------------------------------------------------------------------------------

0847.

0848.RAISERROR('Finding missing FK indexes', 10, 1) WITH NOWAIT

0849.

0850.INSERT INTO #IndexList

0851.(

0852. database\_id

0853. , database\_name

0854. , schema\_id

0855. , schema\_name

0856. , table\_object\_id

0857. , table\_name

0858. , index\_id

0859. , index\_name

0860. , index\_type\_desc

0861. , is\_unique

0862. , rank

0863. , indexed\_columns

0864. , included\_columns

0865.)

0866.SELECT

0867. @DB\_ID AS database\_id

0868. , @DB\_Name AS database\_name

0869. , s.schema\_id AS schema\_id

0870. , s.name AS schema\_name

0871. , t.object\_id AS table\_object\_id

0872. , t.name AS table\_name

0873. , -1 AS index\_id

0874. , fk.fk\_name AS index\_name

0875. , 'MISSING (FOR FK)' AS index\_type\_desc

0876. , 0 AS is\_unique

0877. , 9999 AS rank

0878. , fk.fk\_columns AS indexed\_columns

0879. , '' AS included\_columns

0880. FROM sys.tables t WITH (NOLOCK)

0881. JOIN sys.schemas s WITH (NOLOCK)

0882. ON t.schema\_id = s.schema\_id

0883. JOIN #ForeignKeys fk WITH (NOLOCK)

0884. ON t.object\_id = fk.table\_object\_id

0885. LEFT OUTER JOIN #IndexList il WITH (NOLOCK)

0886. ON fk.table\_object\_id = il.table\_object\_id

0887. AND il.indexed\_columns\_compare LIKE fk.fk\_columns\_compare + '%'

0888. WHERE il.index\_name IS NULL

0889. AND t.is\_ms\_shipped = 0

0890.

0891.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0892.

0893.

0894.-------------------------------------------------------------------------------

0895.-- Mark the rows whose table has a clustered index (i.e., non-heap tables).

0896.-------------------------------------------------------------------------------

0897.

0898.RAISERROR('Marking the indexes whose table has a clustered index', 10, 1) WITH NOWAIT

0899.

0900.UPDATE il

0901. SET table\_has\_clustered = 1

0902. FROM #IndexList il

0903. JOIN (SELECT i.object\_id

0904. FROM sys.indexes i WITH (NOLOCK)

0905. WHERE i.type\_desc = 'CLUSTERED') ui

0906. ON il.table\_object\_id = ui.object\_id

0907.

0908.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0909.

0910.

0911.-------------------------------------------------------------------------------

0912.-- Mark the rows whose table has a LOB.

0913.-------------------------------------------------------------------------------

0914.

0915.RAISERROR('Marking the indexes whose table has a LOB', 10, 1) WITH NOWAIT

0916.

0917.UPDATE il

0918. SET table\_has\_lob = 1

0919. FROM #IndexList il

0920. JOIN (SELECT c.object\_id AS object\_id

0921. FROM sys.columns c WITH (NOLOCK)

0922. WHERE (c.system\_type\_id IN (34, 35, 99) -- 34 = image, 35 = text, 99 = ntext

0923. OR c.max\_length = -1)) lc -- VARBINARY(MAX), VARCHAR(MAX), NVARCHAR(MAX), XML

0924. ON il.table\_object\_id = lc.object\_id

0925.

0926.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0927.

0928.

0929.-------------------------------------------------------------------------------

0930.-- Mark the rows whose table is partitioned.

0931.-------------------------------------------------------------------------------

0932.

0933.RAISERROR('Marking the indexes whose table is partitioned', 10, 1) WITH NOWAIT

0934.

0935.UPDATE il

0936. SET table\_is\_partitioned = CASE COALESCE(lc.mp, 0) WHEN 0 THEN 0 ELSE 1 END

0937. FROM #IndexList il

0938. JOIN (SELECT DISTINCT p.object\_id AS object\_id,

0939. CASE p.partition\_number WHEN 1 THEN 0 ELSE 1 END AS mp

0940. FROM sys.partitions p WITH (NOLOCK)) lc

0941. ON il.table\_object\_id = lc.object\_id

0942.

0943.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0944.

0945.

0946.-------------------------------------------------------------------------------

0947.-- Set a schema recommended action.

0948.-------------------------------------------------------------------------------

0949.

0950.RAISERROR('Determining the recommended index action', 10, 1) WITH NOWAIT

0951.

0952.; WITH IndexAction AS

0953.(

0954. SELECT

0955. id

0956.

0957. , CASE

0958. -- Order is from most important to least important, and pre-requisites first.

0959. WHEN index\_name LIKE '%\_\_[1234567890ABCDEF]' +

0960. '[1234567890ABCDEF]' +

0961. '[1234567890ABCDEF]' +

0962. '[1234567890ABCDEF]' +

0963. '[1234567890ABCDEF]' +

0964. '[1234567890ABCDEF]' +

0965. '[1234567890ABCDEF]' +

0966. '[1234567890ABCDEF]' THEN 'RENAME'

0967. WHEN user\_lookups > user\_seeks

0968. AND index\_type\_desc IN ('CLUSTERED', 'HEAP', 'UNIQUE CLUSTERED') THEN 'REALIGN'

0969. WHEN index\_type\_desc = 'MISSING (FOR FK)' THEN 'CREATE NONCLUSTERED'

0970. WHEN index\_type\_desc = 'MISSING (USAGE)' THEN 'CREATE NONCLUSTERED'

0971. WHEN index\_type\_desc = 'HEAP' THEN 'CREATE CLUSTERED'

0972.

0973. -- Exclude UNIQUE and XML indexes from the subsequent actions, like DROP

0974. -- as they may be enforcing constraints, etc., which we aren't measuring.

0975. WHEN index\_type\_desc = 'XML' THEN ''

0976. WHEN is\_unique = 1 THEN ''

0977.

0978. WHEN index\_type\_desc = 'NONCLUSTERED'

0979. AND ROW\_NUMBER() OVER (PARTITION BY table\_name

0980. ORDER BY user\_total DESC) 1.0 THEN 'CREATE NONCLUSTERED'

0981.

0982. WHEN index\_type\_desc = 'NONCLUSTERED' THEN 'BLEND'

0983.

0984. WHEN ROW\_NUMBER() OVER (PARTITION BY table\_name

0985. ORDER BY user\_total DESC,

0986. rank) > 10 THEN 'DROP'

0987. WHEN user\_total = 0.0 THEN 'DROP'

0988. ELSE ''

0989. END AS index\_action

0990. FROM #IndexList WITH (NOLOCK)

0991.)

0992.UPDATE il

0993. SET index\_action = ia.index\_action

0994. FROM #IndexList il

0995. JOIN IndexAction ia

0996. ON il.id = ia.id

0997.

0998.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

0999.

1000.

1001.-- ============================================================================

1002.-- FRAGMENTATION

1003.--

1004.-- If no schema recommendations made yet, check for fragmentation actions.

1005.-- ============================================================================

1006.

1007.-------------------------------------------------------------------------------

1008.-- INTERNAL FRAGMENTATION

1009.--

1010.-- Internal fragmentation occurs when there is too much free space in the

1011.-- index's leaf pages. That is, when the index pages are not filled to the

1012.-- initial (or, desired) fill factor level. On the other hand, we don't want

1013.-- the pages to become too full, or page splits will result (for non-heap

1014.-- tables only, where row order matters). REORGANIZE will compact an index by

1015.-- trying to fill pages to the FILLFACTOR that was last specified. Note that

1016.-- REORGANIZE cannot reduce page fullness: an index rebuild must be performed

1017.-- instead. "F" is the value of the fill factor for this index.

1018.-------------------------------------------------------------------------------

1019.

1020.-- 0 .................. avg\_page\_space\_used\_in\_percent ................... 100

1021.--

1022.-- F

1023.--

1024.-- |--------------- Internal Fragmentation ----------------|

1025.--

1026.-- |------------- REBUILD ---------------|--- REORGANIZE ---|

1027.--

1028.-- ...........................................................................

1029.

1030.RAISERROR('Looking for internal fragmentation', 10, 1) WITH NOWAIT

1031.

1032.UPDATE #IndexList

1033. SET index\_action =

1034. CASE

1035. WHEN il.avg\_page\_space\_used\_in\_percent < @nPageFullRebuildValue THEN 'REBUILD'

1036. WHEN il.avg\_page\_space\_used\_in\_percent < @nPageFullReorgValue THEN 'REORGANIZE'

1037. ELSE ''

1038. END

1039. FROM #IndexList il

1040. WHERE il.index\_action = ''

1041. AND il.disk\_page\_count > @nSmallestTableToProcessInPages -- Ignore small tables.

1042. AND il.table\_has\_clustered = 1 -- Exclude heaps.

1043.

1044.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1045.

1046.

1047.-------------------------------------------------------------------------------

1048.-- LOGICAL/EXTERNAL FRAGMENTATION

1049.--

1050.-- External, aka logical, fragmentation occurs when the logical ordering of

1051.-- the index does not match the physical ordering of the index pages. The

1052.-- value for avg\_fragmentation\_in\_percent should be as close to zero as

1053.-- possible for maximum performance. However, values from 0 percent through 10

1054.-- percent may be acceptable. External fragmentation isn’t too big of a deal

1055.-- for specific searches that return very few records, or queries that return

1056.-- result sets that do not need to be ordered. It's bad for long scans though.

1057.-------------------------------------------------------------------------------

1058.

1059.-- 0 .................... avg\_fragmentation\_in\_percent ................... 100

1060.--

1061.--

1062.-- |-------------------- Logical Fragmentation ------------------------|

1063.--

1064.-- |--- REORGANIZE ---|---------------------------- REBUILD -----------|

1065.--

1066.--.............................................................................

1067.

1068.RAISERROR('Looking for any logical/external fragmentation', 10, 1) WITH NOWAIT

1069.

1070.UPDATE #IndexList

1071. SET index\_action =

1072. CASE

1073. WHEN il.avg\_fragmentation\_in\_percent > @nExtFragRebuildValue THEN 'REBUILD'

1074. WHEN il.avg\_fragmentation\_in\_percent > @nExtFragReorgValue THEN 'REORGANIZE'

1075. ELSE ''

1076. END

1077. FROM #IndexList il

1078. WHERE il.index\_action = ''

1079. AND il.disk\_page\_count > @nSmallestTableToProcessInPages -- Ignore small tables.

1080. AND il.table\_has\_clustered = 1 -- Non-heaps only.

1081.

1082.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1083.

1084.

1085.-------------------------------------------------------------------------------

1086.-- EXTENT FRAGMENTATION

1087.--

1088.-- Extent Fragmentation is the percentage of out-of-order extents in the leaf

1089.-- pages of a heap. An out-of-order extent is one for which the extent that

1090.-- contains the current page for a heap isn't physically the next extent after

1091.-- the extent that contains the previous page. The easiest way to fix this is

1092.-- to cluster the table (and then un-cluster it if so desired).

1093.-------------------------------------------------------------------------------

1094.

1095.RAISERROR('Looking for any heap tables'' extent fragmentation', 10, 1) WITH NOWAIT

1096.

1097.UPDATE #IndexList

1098. SET index\_action =

1099. CASE

1100. WHEN il.avg\_fragmentation\_in\_percent > @nExtFragReorgValue THEN 'CLUSTER'

1101. ELSE ''

1102. END

1103. FROM #IndexList il

1104. WHERE il.index\_action = ''

1105. AND il.disk\_page\_count > @nSmallestTableToProcessInPages -- Ignore small tables.

1106. AND il.table\_has\_clustered = 0 -- Heaps only.

1107. AND il.table\_has\_lob = 0

1108.

1109.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1110.

1111.

1112.-------------------------------------------------------------------------------

1113.-- If no recommendations made yet, check for statistics actions.

1114.-------------------------------------------------------------------------------

1115.

1116.RAISERROR('Determining the recommended statistics action', 10, 1) WITH NOWAIT

1117.

1118.UPDATE il

1119. SET index\_action =

1120. CASE

1121. WHEN date\_stats\_updated IS NULL THEN 'CREATE STATS'

1122. WHEN ssi.rowmodctr > il.row\_stat\_threshold THEN 'UPDATE STATS'

1123. WHEN DATEDIFF(d, date\_stats\_updated, GETDATE()) > 90 THEN 'UPDATE STATS'

1124. ELSE ''

1125. END

1126. FROM #IndexList il

1127. JOIN sys.sysindexes ssi

1128. ON il.table\_object\_id = ssi.id

1129. WHERE index\_action = ''

1130.

1131.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1132.

1133.

1134.-------------------------------------------------------------------------------

1135.-- Remove \*all\* the rows that have no action recommended.

1136.-------------------------------------------------------------------------------

1137.

1138.RAISERROR('Excluding indexes with no recommended action', 10, 1) WITH NOWAIT

1139.

1140.DELETE FROM #IndexList

1141. WHERE index\_action = ''

1142.

1143.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1144.

1145.

1146.-------------------------------------------------------------------------------

1147.-- Mark the rows whose table has a unique index. Just informational.

1148.-------------------------------------------------------------------------------

1149.

1150.RAISERROR('Marking the indexes whose table has a unique index', 10, 1) WITH NOWAIT

1151.

1152.UPDATE il

1153. SET table\_has\_unique = 1

1154. FROM #IndexList il

1155. JOIN (SELECT DISTINCT object\_id

1156. FROM sys.indexes i WITH (NOLOCK)

1157. WHERE i.is\_unique = 1) ui

1158. ON il.table\_object\_id = ui.object\_id

1159.

1160.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1161.

1162.

1163.-------------------------------------------------------------------------------

1164.-- Set the recommended action T-SQL code. Not all the recommended actions

1165.-- require a USE command.

1166.-------------------------------------------------------------------------------

1167.

1168.RAISERROR('Creating the DML for the recommended actions', 10, 1) WITH NOWAIT

1169.

1170.UPDATE #IndexList

1171. SET dml\_for\_action = '/\* ' + index\_action + REPLICATE(' ', 19 - (DATALENGTH(index\_action) / 2)) + ' \*/ ' +

1172. CASE index\_action

1173. WHEN 'BLEND' THEN ''

1174. WHEN 'CREATE CLUSTERED' THEN 'USE ' + @DB\_Name + ' '

1175. WHEN 'CREATE NONCLUSTERED' THEN 'USE ' + @DB\_Name + ' '

1176. WHEN 'CREATE STATS' THEN 'USE ' + @DB\_Name + ' '

1177. WHEN 'DROP' THEN 'USE ' + @DB\_Name + ' '

1178. WHEN 'RENAME' THEN 'USE ' + @DB\_Name + ' '

1179. WHEN 'REALIGN' THEN ''

1180. WHEN 'REBUILD' THEN 'USE ' + @DB\_Name + ' '

1181. WHEN 'REORGANIZE' THEN 'USE ' + @DB\_Name + ' '

1182. WHEN 'UPDATE STATS' THEN 'USE ' + @DB\_Name + ' '

1183. ELSE NULL

1184. END

1185.

1186.UPDATE #IndexList

1187. SET dml\_for\_action +=

1188. CASE index\_action

1189.

1190. -- Review the missing index details to see if the missing index details

1191. -- can be added to an existing index, for example changing a non-clustered

1192. -- index to a clustered one, or adding a unique index.

1193. WHEN 'BLEND' THEN

1194. ' -- See if this missing index can be added to an existing index: ' + index\_name

1195.

1196. -- Add a clustered index to this heap table.

1197. WHEN 'CREATE CLUSTERED' THEN

1198. 'IF NOT EXISTS (SELECT \* FROM sys.indexes '

1199. + 'WHERE name = ''PK\_' + table\_name + '\_<column\_for\_clustered\_index\_on\_table\_' + table\_name + ', NVARCHAR(MAX), ThisColumn> '' '

1200. + 'AND index\_type\_desc = ''CLUSTERED'') '

1201. + 'BEGIN '

1202. + 'RAISERROR(''Creating the clustered index PK\_' + table\_name + '<column\_for\_clustered\_index\_on\_table\_' + table\_name + ', NVARCHAR(MAX), ThisColumn>'', 10, 1) WITH NOWAIT, LOG '

1203. + ' '

1204. + 'CREATE UNIQUE CLUSTERED INDEX PK\_' + table\_name + '\_<column\_for\_clustered\_index\_on\_table\_' + table\_name + ', NVARCHAR(MAX), ThisColumn> '

1205. + 'ON ' + schema\_name + '.' + table\_name + ' '

1206. + '(<column\_for\_clustered\_index\_on\_table\_' + table\_name + ', NVARCHAR(MAX), ThisColumn>) '

1207. + 'WITH (PAD\_INDEX = ON, FILLFACTOR = ' + @sFillFactor + ') '

1208. + 'END '

1209.

1210. -- Add the non-clustered index to the table.

1211. WHEN 'CREATE NONCLUSTERED' THEN

1212. 'IF NOT EXISTS (SELECT \* FROM sys.indexes '

1213. + 'WHERE name = ''' + REPLACE(index\_name, ' | REFERENCING', '') + ''''

1214. + CASE index\_type\_desc

1215. WHEN 'MISSING (FOR FK)' THEN ''

1216. WHEN 'MISSING (USAGE)' THEN ''

1217. WHEN 'HEAP' THEN ''

1218. ELSE 'AND index\_type\_desc = ''' + index\_type\_desc + ''''

1219. END

1220. + ') '

1221. + 'BEGIN '

1222. + 'RAISERROR(''Creating the index ' + REPLACE(index\_name, ' | REFERENCING', '') + ''', 10, 1) WITH NOWAIT, LOG '

1223. + ' '

1224. + 'CREATE NONCLUSTERED '

1225. + CASE is\_unique WHEN 1 THEN 'UNIQUE' ELSE '' END

1226.

1227. + CASE index\_type\_desc

1228. WHEN 'MISSING (FOR FK)' THEN ''

1229. WHEN 'MISSING (USAGE)' THEN ''

1230. WHEN 'HEAP' THEN ''

1231. ELSE index\_type\_desc

1232. END

1233.

1234. + 'INDEX ' + REPLACE(index\_name, ' | REFERENCING', '') + ' '

1235. + 'ON ' + schema\_name + '.' + QUOTENAME(table\_name)

1236. + '(' + indexed\_columns + ') '

1237. + CASE included\_columns

1238. WHEN '' THEN ''

1239. ELSE 'INCLUDED (' + included\_columns + ') '

1240. END

1241. + 'WITH (PAD\_INDEX = ON, FILLFACTOR = ' + @sFillFactor + ') '

1242. + 'END '

1243.

1244. -- Create missing or NULL statistics.

1245. WHEN 'CREATE STATS' THEN

1246. 'CREATE STATISTICS ' + index\_name + ' ON ' + table\_name + '(' + indexed\_columns + ') WITH FULLSCAN'

1247.

1248. -- Drop the index from the table.

1249. WHEN 'DROP' THEN

1250. 'IF EXISTS (SELECT \* FROM sys.indexes '

1251. + 'WHERE name = ''' + REPLACE(index\_name, ' | REFERENCING', '') + ''' '

1252. + CASE index\_type\_desc

1253. WHEN 'MISSING (FOR FK)' THEN ''

1254. WHEN 'MISSING (USAGE)' THEN ''

1255. WHEN 'HEAP' THEN ''

1256. ELSE 'AND index\_type\_desc = ''' + index\_type\_desc + ''' '

1257. END

1258. + ') '

1259. + 'BEGIN '

1260. + 'RAISERROR(''Dropping the index ' + REPLACE(index\_name, ' | REFERENCING', '') + ''', 10, 1) WITH NOWAIT, LOG '

1261. + ' '

1262. + 'DROP INDEX ' + REPLACE(index\_name, ' | REFERENCING', '') + ' '

1263. + 'ON ' + schema\_name + '.' + QUOTENAME(table\_name) + ' '

1264. + 'END '

1265.

1266. -- Investigate whether to move the clustered index to another index,

1267. -- or add included columns to the indexes that are part of the

1268. -- bookmark lookups.

1269. WHEN 'REALIGN' THEN

1270. ' -- See whether to move this clustered index to another index, ' +

1271. 'or add included columns to the indexes that are part of the bookmark lookups: ' + index\_name

1272.

1273. -- A system-generated name usually looks like PK\_\_tablename\_\_0F1E2D. We rename this

1274. -- to use the name of the index's first column, and fix the underscores:

1275. -- PK\_tablename\_columnname .

1276. WHEN 'RENAME' THEN

1277. 'DECLARE @sNewIndexName sysname '

1278.

1279. + 'SELECT @sNewIndexName = ''PK\_'' + OBJECT\_NAME(i.object\_id) + ''\_'' + c.name '

1280. + 'FROM sys.indexes i '

1281. + 'JOIN sys.index\_columns ic '

1282. + 'ON i.object\_id = ic.object\_id '

1283. + 'JOIN sys.columns c '

1284. + 'ON i.object\_id = c.object\_id '

1285. + 'AND ic.column\_id = c.column\_id '

1286. + 'WHERE OBJECT\_NAME(i.object\_id) = ''' + table\_name + ''' '

1287. + 'AND i.name = ''' + index\_name + ''' '

1288. + 'AND ic.index\_column\_id = 1 '

1289.

1290. + 'DECLARE @sOldSchemaTableIndex sysname '

1291. + 'SET @sOldSchemaTableIndex = ''' + schema\_name + '.' + table\_name + '.' + index\_name + ''' '

1292. + 'EXECUTE sp\_rename @sOldSchemaTableIndex, @sNewIndexName, ''INDEX'' '

1293.

1294. -- Drops and recreates the index; eliminates fragmentation.

1295. -- Index rebuilds automatically update statistics with a full scan. They

1296. -- also reclaim disk space, and reorder the index rows into contiguous pages.

1297. -- As SQL Server creates a new copy of the index before removing the old one,

1298. -- it requires free space in the data files approximately equivalent to the

1299. -- size of the index. The index rebuilding process uses more CPU, and it

1300. -- locks the database resources.

1301. --

1302. -- Rebuild can be done ONLINE (with the proper editions) except for:

1303. -- o Clustered indexes with LOB data

1304. -- o Non-clustered indexes that includes a column with LOB data

1305. -- o Partitioned indexes (because the entire table will be locked during the rebuild)

1306. WHEN 'REBUILD' THEN

1307. 'ALTER INDEX ' + index\_name + ' ON ' + table\_name + ' ' +

1308. 'REBUILD WITH (FILLFACTOR = ' + @sFillFactor + ', ONLINE = ' +

1309. CASE

1310. WHEN table\_has\_lob = 0 THEN @sOnlineRebuild

1311. WHEN table\_is\_partitioned = 0 THEN @sOnlineRebuild

1312. ELSE 'NO'

1313. END + ')'

1314.

1315. -- Reduces external fragmentation by physically reordering the

1316. -- leaf-level pages to match the logical, left to right, order of the leaf

1317. -- nodes. It also reduces internal fragmentation by compacting the index

1318. -- pages based on the existing fill factor value. Requires only one page of

1319. -- additional space to run, and it is always performed ONLINE. The biggest

1320. -- advantage is that it does not require locks for long times, so it doesn't

1321. -- block updates or other user queries. Reorganizing an index does not

1322. -- update statistics, so we do that manually.

1323. WHEN 'REORGANIZE' THEN

1324. 'ALTER INDEX ' + index\_name + ' ON ' + table\_name + ' REORGANIZE'

1325.

1326. -- Update existing, stale statistics.

1327. WHEN 'UPDATE STATS' THEN

1328. 'UPDATE STATISTICS ' + table\_name + ' ' + index\_name + ' WITH FULLSCAN '

1329.

1330. -- Programming error.

1331. ELSE

1332. ' \*\*\* ERROR: Unknown index action ' + index\_action

1333.END

1334.

1335.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1336.

1337.

1338.-------------------------------------------------------------------------------

1339.-- Check output for validity.

1340.-------------------------------------------------------------------------------

1341.

1342.IF EXISTS (SELECT \* FROM #IndexList WITH (NOLOCK) WHERE dml\_for\_action IS NULL)

1343.BEGIN

1344. RAISERROR(' $$$ dml\_for\_action IS NULL $$$', 16, 1) WITH NOWAIT, LOG

1345.

1346. SELECT id,

1347. index\_type\_desc,

1348. index\_action,

1349. index\_name

1350. FROM #IndexList WITH (NOLOCK)

1351. WHERE dml\_for\_action IS NULL

1352.

1353. RAISERROR(' $$$ See results grid for incorrect rows $$$', 20, 1) WITH NOWAIT, LOG

1354.END

1355.

1356.

1357.-------------------------------------------------------------------------------

1358.-- Send final results to grid.

1359.-------------------------------------------------------------------------------

1360.

1361.RAISERROR('Sending final results to grid', 10, 1) WITH NOWAIT

1362.

1363.IF 1=0

1364. -- This sends the full results to the grid.

1365. SELECT \*

1366. FROM #IndexList WITH (NOLOCK)

1367. ORDER BY database\_name, schema\_name, table\_name, index\_id, partition\_number

1368. --ORDER BY rank DESC,

1369. -- impact DESC,

1370. -- table\_name

1371.ELSE

1372. -- This sends the important columns to the grid.

1373. SELECT rank

1374. , impact

1375. , table\_name

1376. , index\_name

1377. , index\_action

1378. , index\_type\_desc

1379. , indexed\_columns

1380. , included\_columns

1381. , row\_count

1382. , avg\_fragmentation\_in\_percent

1383. , avg\_page\_space\_used\_in\_percent

1384. , dml\_for\_action

1385. FROM #IndexList WITH (NOLOCK)

1386. ORDER BY rank DESC,

1387. impact DESC,

1388. table\_name

1389.

1390.RAISERROR(' Processed %d row(s)', 10, 1, @@ROWCOUNT) WITH NOWAIT

1391.

1392.

1393.-------------------------------------------------------------------------------

1394.-- Print final results to screen.

1395.-------------------------------------------------------------------------------

1396.

1397.RAISERROR('Printing final results to screen', 10, 1) WITH NOWAIT

1398.

1399.PRINT ' '

1400.PRINT '|---------------------------- snip ----------------------------------->'

1401.PRINT ' '

1402.PRINT 'BEGIN TRAN'

1403.PRINT ' '

1404.

1405.DECLARE @sDML NVARCHAR(4000) = ''

1406.

1407.DECLARE TheCursor CURSOR FOR

1408. SELECT dml\_for\_action FROM #IndexList WITH (NOLOCK)

1409.

1410.OPEN TheCursor

1411.

1412.FETCH NEXT FROM TheCursor

1413. INTO @sDML

1414.

1415.WHILE @@FETCH\_STATUS = 0

1416.BEGIN

1417. PRINT @sDML

1418. FETCH NEXT FROM TheCursor INTO @sDML

1419.END

1420.

1421.CLOSE TheCursor

1422.DEALLOCATE TheCursor

1423.

1424.PRINT ' '

1425.PRINT 'ROLLBACK -- COMMIT'

1426.PRINT ' '

1427.PRINT '<---------------------------- snip ------------------------------------|'

1428.

1429.

1430.-------------------------------------------------------------------------------

1431.-- Done.

1432.-------------------------------------------------------------------------------

1433.

1434.PRINT ' '

1435.PRINT 'Done.'

1436.

1437.

1438.-------------------------------------------------------------------------------

1439.-- Based on original work at:

1440.--

1441.-- [http://indexanalysis.codeplex.com/Wiki/View.aspx?ProjectName=IndexAnalysis](http://indexanalysis.codeplex.com/Wiki/View.aspx?ProjectName=IndexAnalysis%20) ;

1442.--

1443.-- Microsoft Public License (Ms-PL)

1444.-- ================================

1445.-- This license governs use of the accompanying software. If you use the

1446.-- software, you accept this license. If you do not accept the license, do not

1447.-- use the software.

1448.--

1449.-- 1. Definitions

1450.--

1451.-- The terms "reproduce," "reproduction," "derivative works," and

1452.-- "distribution" have the same meaning here as under U.S. copyright law.

1453.--

1454.-- A "contribution" is the original software, or any additions or changes

1455.-- to the software.

1456.--

1457.-- A "contributor" is any person that distributes its contribution under

1458.-- this license.

1459.--

1460.-- "Licensed patents" are a contributor's patent claims that read directly

1461.-- on its contribution.

1462.--

1463.-- 2. Grant of Rights

1464.--

1465.-- (A) Copyright Grant- Subject to the terms of this license, including the

1466.-- license conditions and limitations in section 3, each contributor

1467.-- grants you a non-exclusive, worldwide, royalty-free copyright license

1468.-- to reproduce its contribution, prepare derivative works of its

1469.-- contribution, and distribute its contribution or any derivative works

1470.-- that you create.

1471.--

1472.-- (B) Patent Grant- Subject to the terms of this license, including the

1473.-- license conditions and limitations in section 3, each contributor

1474.-- grants you a non-exclusive, worldwide, royalty-free license under

1475.-- its licensed patents to make, have made, use, sell, offer for sale,

1476.-- import, and/or otherwise dispose of its contribution in the software

1477.-- or derivative works of the contribution in the software.

1478.--

1479.-- 3. Conditions and Limitations

1480.--

1481.-- (A) No Trademark License- This license does not grant you rights to use

1482.-- any contributors' name, logo, or trademarks.

1483.--

1484.-- (B) If you bring a patent claim against any contributor over patents that

1485.-- you claim are infringed by the software, your patent license from

1486.-- such contributor to the software ends automatically.

1487.--

1488.-- (C) If you distribute any portion of the software, you must retain all

1489.-- copyright, patent, trademark, and attribution notices that are

1490.-- present in the software.

1491.--

1492.-- (D) If you distribute any portion of the software in source code form,

1493.-- you may do so only under this license by including a complete copy

1494.-- of this license with your distribution. If you distribute any portion

1495.-- of the software in compiled or object code form, you may only do so

1496.-- under a license that complies with this license.

1497.--

1498.-- (E) The software is licensed "as-is." You bear the risk of using it. The

1499.-- contributors give no express warranties, guarantees or conditions.

1500.-- You may have additional consumer rights under your local laws which

1501.-- this license cannot change. To the extent permitted under your local

1502.-- laws, the contributors exclude the implied warranties of

1503.-- merchantability, fitness for a particular purpose and

1504.-- non-infringement.

1505.-------------------------------------------------------------------------------