Understanding and Managing SQL Server Fragmentation

Greg Robidoux
Edgewood Solutions
gregr@edgewoodsolutions.com

Bulleit Manale
Idera
www.idera.com
Idera Solutions
for SQL Server

Performance & Availability

- **SQL diagnostic manager™**
  Monitor, diagnose & manage performance

- **SQL mobile manager™**
  Monitor, diagnose & manage performance from a PDA

- **SQL defrag manager™**
  Automate & optimize database defragmentation

Backup & Recovery

- **SQL safe™**
  Accelerate, compress & encrypt database backups

Compliance & Security

- **SQL compliance manager™**
  Capture, audit & alert on user activity

- **SQL secure™**
  Assess security risks & audit access rights

Change Management

- **SQL change manager™**
  Manage & monitor schema changes

Administration

- **SQL admin toolset™**
  24 essential tools for monitoring, administering, troubleshooting and reporting

Idera
MANAGING YOUR WINDOWS WORLD™
Agenda - Fragmentation

- Overview
  - Understanding Storage
  - What fragmentation is and how it occurs
  - Detecting
  - Fixing
  - Managing
- Other Tools
- Questions / Wrap Up
Storage

• Data storage
  • Pages - tables and indexes are stored as a collection of 8-KB pages
  • Extents – 8 contiguous pages = 64KB total space

• Cluster or Heap
  ◦ Clustered – data is stored in order based on the clustered index (b-tree)
  ◦ Heap – there is no particular order to how data is stored

• Partitions
  ◦ Tables can have one or more partitions. The default is one per table.
Storage

## Storage

<table>
<thead>
<tr>
<th>Allocation type</th>
<th>Used to manage</th>
</tr>
</thead>
</table>
| **IN_ROW_DATA (Data)**       | • Data or index rows that contain all data, except large object (LOB) data.  
                                | • Pages are of type Data or Index.                                                                                                           |
| **LOB_DATA (LOB)**           | • Large object data stored in one or more of these data types: text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max), or CLR user-defined types (CLR UDT).  
                                | • Pages are of type Text/Image                                                                                                               |
| **ROW_OVERFLOW_DATA (Row-Overflow)** | • Variable length data stored in varchar, nvarchar, varbinary, or sql_variant columns that exceed the 8,060 byte row size limit.  
                                | • Pages are of type Data                                                                                                                     |
Storage - HEAP

- A heap is a table without a clustered index
- Uses Index Allocation Map (IAM) pages to find data
Storage - HEAP

Storage - Indexes

- Indexes
  - Clustered
  - Non-clustered
  - XML
  - Full Text

- Clustered and Non-clustered
  - Use b-tree index structure
Storage – B-TREE

- B-TREE Structure
  - In SQL Server, indexes are organized as B-trees.
  - Each page in an index B-tree is called an index node.
  - The top node of the B-tree is called the root node.
  - The bottom level of nodes in the index is called the leaf nodes.
  - Any index levels between the root and the leaf nodes are collectively known as intermediate levels.
  - In a clustered index, the leaf nodes contain the data pages of the underlying table.
  - The root and leaf nodes contain index pages holding index rows.
  - Each index row contains a key value and a pointer to either an intermediate level page in the B-tree, or a data row in the leaf level of the index.
  - The pages in each level of the index are linked in a doubly-linked list. This means that there are pointers to the previous and next pages.
Storage - Clustered Index

Storage - Non-Clustered Index

Storage - Allocation

- `sys.allocation_units` - contains a row for each allocation unit in the database.

```sql
SELECT o.name AS table_name, p.index_id, i.name AS index_name, au.type_desc AS allocation_type, au.data_pages,...
FROM sys.allocation_units AS au
JOIN sys.partitions AS p ON au.container_id = p.partition_id
JOIN sys.objects AS o ON p.object_id = o.object_id
JOIN sys.indexes AS i ON p.index_id = i.index_id
AND i.object_id = p.object_id
```
# Storage - Allocation

<table>
<thead>
<tr>
<th>table_name</th>
<th>index_id</th>
<th>index_name</th>
<th>allocation_type</th>
<th>data_pages</th>
<th>partition_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesOrderDetail</td>
<td>1</td>
<td>FK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID</td>
<td>IN_ROW_DATA</td>
<td>1233</td>
<td>1</td>
</tr>
<tr>
<td>SalesOrderDetail</td>
<td>2</td>
<td>AK_SalesOrderDetail_rowguid</td>
<td>IN_ROW_DATA</td>
<td>406</td>
<td>1</td>
</tr>
<tr>
<td>SalesOrderDetail</td>
<td>3</td>
<td>IX_SalesOrderDetail_ProductID</td>
<td>IN_ROW_DATA</td>
<td>226</td>
<td>1</td>
</tr>
<tr>
<td>CurrencyRate</td>
<td>1</td>
<td>FK_CurrencyRate_CurrencyRateID</td>
<td>IN_ROW_DATA</td>
<td>96</td>
<td>1</td>
</tr>
<tr>
<td>CurrencyRate</td>
<td>2</td>
<td>AK_CurrencyRate_CurrencyRateID_Date_FromCurrencyCode_ToCurrencyCode</td>
<td>IN_ROW_DATA</td>
<td>46</td>
<td>1</td>
</tr>
<tr>
<td>Customer</td>
<td>1</td>
<td>FK_Customer_CustomerID</td>
<td>IN_ROW_DATA</td>
<td>103</td>
<td>1</td>
</tr>
<tr>
<td>Customer</td>
<td>2</td>
<td>AK_Customer_rowguid</td>
<td>IN_ROW_DATA</td>
<td>55</td>
<td>1</td>
</tr>
<tr>
<td>Customer</td>
<td>3</td>
<td>AK_Customer_AccountNumber</td>
<td>IN_ROW_DATA</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>Customer</td>
<td>5</td>
<td>IX_Customer_TerritoryID</td>
<td>IN_ROW_DATA</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>1</td>
<td>FK_SalesOrderHeader_SalesOrderID</td>
<td>IN_ROW_DATA</td>
<td>639</td>
<td>1</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>2</td>
<td>AK_SalesOrderHeader_rowguid</td>
<td>IN_ROW_DATA</td>
<td>90</td>
<td>1</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>3</td>
<td>AK_SalesOrderHeader_SalesOrderNumber</td>
<td>IN_ROW_DATA</td>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>5</td>
<td>IX_SalesOrderHeader_CustomerID</td>
<td>IN_ROW_DATA</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>6</td>
<td>IX_SalesOrderHeader_SalesPersonID</td>
<td>IN_ROW_DATA</td>
<td>55</td>
<td>1</td>
</tr>
<tr>
<td>CustomerAddress</td>
<td>1</td>
<td>FK_CustomerAddress_CustomerID_AddressID</td>
<td>IN_ROW_DATA</td>
<td>108</td>
<td>1</td>
</tr>
<tr>
<td>CustomerAddress</td>
<td>2</td>
<td>AK_CustomerAddress_rowguid</td>
<td>IN_ROW_DATA</td>
<td>65</td>
<td>1</td>
</tr>
<tr>
<td>Document</td>
<td>1</td>
<td>FK_Document_DocumentID</td>
<td>LOB_DATA</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Document</td>
<td>1</td>
<td>FK_Document_DocumentID</td>
<td>IN_ROW_DATA</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Document</td>
<td>1</td>
<td>FK_Document_DocumentID</td>
<td>ROW_OVERFLOW_DATA</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Document</td>
<td>2</td>
<td>AK_Document_FileName_Revision</td>
<td>IN_ROW_DATA</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
What is fragmentation

What it is

- **Fragmentation** is when storage space is used inefficiently, reducing storage capacity and in most cases performance.
- **Internal fragmentation** occurs when storage is allocated without using it.
- **External fragmentation** is when storage becomes divided into many small pieces over time.
What is fragmentation

- **SQL Server Fragmentation**
  - **Logical Fragmentation**
    - This is the percentage of out-of-order pages in the leaf pages of an index.
    - An out-of-order page is one for which the next page indicated in an IAM is a page that is different from the page pointed to by the next page pointer in the leaf page.
  - **Extent Fragmentation**
    - This is the percentage of out-of-order extents in the leaf pages of a heap.
    - An out-of-order extent is one for which the extent that contains the current page for a heap is not physically the next extent after the extent that contains the previous page.
Logical Fragmentation

Page level fragmentation

Diagram of page level fragmentation showing segments of memory.
Extent Fragmentation

Fragmentation – before

Fragmentation – after
When it happens

• How it happens
  ◦ Data modification
    • Inserting – causes page splits
    • Deleting – leaves free space
    • Updating – changing index values
  ◦ Shrinking
    • DBCC SHRINKDATABASE
    • DBCC SHRINKFILE (data files)
    • AutoShrink
Why is this bad

- Uses additional space
- Increased I/O for range scans
- Impacts performance
- Places additional demands on hardware
Detecting

- **DBCC SHOWCONTIG**
- **sys.dm_db_index_physical_stats**
  - **LIMITED** - fastest and scans the smallest number of pages. It scans all pages for a heap, but only the parent-level pages for an index, which are the pages above the leaf-level.
  - **SAMPLED** - returns statistics based on a one percent sample of all the pages in the index or heap. If the index or heap has fewer than 10,000 pages DETAILED mode is used.
  - **DETAILED** - scans all pages and returns all statistics. This also takes the longest
Detecting

SELECT *
FROM sys.dm_db_index_physical_stats (null)
   (null)
   (null)
   (null)
   (null)
   (null)
   (null)
   (null)
   (null)
   (null)

22
Detecting – all DBs all indexes

SELECT *
FROM sys.dm_db_index_physical_stats ( NULL, NULL, NULL, NULL, NULL, NULL, 'LIMITED');
DECLARE @db_id SMALLINT;
DECLARE @object_id INT;
SET @db_id = DB_ID(N'Test');
SET @object_id = OBJECT_ID(N'Test.dbo.Address');

SELECT *
FROM sys.dm_db_index_physical_stats ( 
    @db_id,
    @object_id,
    NULL,
    NULL ,
    'LIMITED');
## DMF - Output

<table>
<thead>
<tr>
<th>DBName</th>
<th>TName</th>
<th>index_id</th>
<th>partition_number</th>
<th>index_type_desc</th>
<th>alloc_unit_type_desc</th>
<th>index_depth</th>
<th>index_level</th>
<th>avg_fragmentation_in_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>1</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>2</td>
<td>0</td>
<td>28.57142857</td>
</tr>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>1</td>
<td>1</td>
<td>CLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>2</td>
<td>1</td>
<td>NONCLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>2</td>
<td>0</td>
<td>66.66666667</td>
</tr>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>2</td>
<td>1</td>
<td>NONCLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>3</td>
<td>1</td>
<td>NONCLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>2</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>3</td>
<td>1</td>
<td>NONCLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>4</td>
<td>1</td>
<td>NONCLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AdventureWorks</td>
<td>Employee</td>
<td>4</td>
<td>1</td>
<td>NONCLUSTERED INDEX</td>
<td>IN_ROW_DATA</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fragment_count</th>
<th>avg_fragment_size_in_pages</th>
<th>page_count</th>
<th>avg_page_space_used_in_percent</th>
<th>record_count</th>
<th>ghost_record_count</th>
<th>version_ghost_record_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1.75</td>
<td>7</td>
<td>97.5879145</td>
<td>290</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.096579936</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>66.58430936</td>
<td>290</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2.112876056</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>51.32196689</td>
<td>290</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0.716580183</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0.23201137</td>
<td>290</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>0.03590314</td>
<td>290</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>min_record_size_in_bytes</th>
<th>max_record_size_in_bytes</th>
<th>avg_record_size_in_bytes</th>
<th>forwarded_record_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>143</td>
<td>219</td>
<td>189.255</td>
<td>NULL</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>NULL</td>
</tr>
<tr>
<td>47</td>
<td>65</td>
<td>53.772</td>
<td>NULL</td>
</tr>
<tr>
<td>53</td>
<td>59</td>
<td>55.866</td>
<td>NULL</td>
</tr>
<tr>
<td>19</td>
<td>27</td>
<td>26.662</td>
<td>NULL</td>
</tr>
<tr>
<td>27</td>
<td>25</td>
<td>28</td>
<td>NULL</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>21</td>
<td>NULL</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>NULL</td>
</tr>
</tbody>
</table>
SHOWCONTIG - Output

<table>
<thead>
<tr>
<th>DBCC SHOWCONTIG scanning 'Employee' table...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table: 'Employee' (869578136); index ID: 1, database ID: 10</td>
</tr>
<tr>
<td>TABLE level scan performed.</td>
</tr>
<tr>
<td>- Pages Scanned: 7</td>
</tr>
<tr>
<td>- Extents Scanned: 3</td>
</tr>
<tr>
<td>- Extent Switches: 3</td>
</tr>
<tr>
<td>- Avg. Pages per Extent: 2.3</td>
</tr>
<tr>
<td>- Scan Density [Best Count:Actual Count]: 25.00% [1:4]</td>
</tr>
<tr>
<td>- Logical Scan Fragmentation: 28.57%</td>
</tr>
<tr>
<td>- Extent Scan Fragmentation: 33.33%</td>
</tr>
<tr>
<td>- Avg. Bytes Free per Page: 172.6</td>
</tr>
<tr>
<td>- Avg. Page Density (full): 97.87%</td>
</tr>
</tbody>
</table>

DBCC execution completed. If DBCC printed error messages, contact your system administrator.
# sys.dm_db_index_physical_stats

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database_id</code></td>
<td>Database ID of the table or view.</td>
</tr>
<tr>
<td><code>object_id</code></td>
<td>Object ID of the table or view that the index is on.</td>
</tr>
<tr>
<td><code>index_id</code></td>
<td>Index ID of an index. 0 = HEAP</td>
</tr>
<tr>
<td><code>partition_number</code></td>
<td>Partition number.</td>
</tr>
<tr>
<td><code>index_type_desc</code></td>
<td>Description of the index type: HEAP, Clustered, Non-clustered, etc…</td>
</tr>
<tr>
<td><code>alloc_unit_type_desc</code></td>
<td>Description of the allocation unit type: In-Row, LOB, Overflow</td>
</tr>
<tr>
<td><code>index_depth</code></td>
<td>Number of index levels.</td>
</tr>
<tr>
<td><code>index_level</code></td>
<td>Current level of the index.</td>
</tr>
<tr>
<td><code>avg_fragmentation_in_percent</code></td>
<td>Logical fragmentation for indexes, or extent fragmentation for heaps in the IN_ROW_DATA allocation unit.</td>
</tr>
<tr>
<td><code>fragment_count</code></td>
<td>Number of fragments in the leaf level of an IN_ROW_DATA allocation unit.</td>
</tr>
<tr>
<td><code>avg_fragment_size_in_pages</code></td>
<td>Average number of pages in one fragment in the leaf level of an IN_ROW_DATA allocation unit.</td>
</tr>
<tr>
<td><code>page_count</code></td>
<td>Total number of index or data pages.</td>
</tr>
<tr>
<td><code>avg_page_space_used_in_percent</code></td>
<td>Average percentage of available data storage space used in all pages.</td>
</tr>
<tr>
<td><code>record_count</code></td>
<td>Total number of records.</td>
</tr>
<tr>
<td><code>ghost_record_count</code></td>
<td>Number of deleted records (ghost records) ready for removal by the ghost cleanup task in the allocation unit.</td>
</tr>
<tr>
<td><code>version_ghost_record_count</code></td>
<td>Number of ghost records retained by an outstanding snapshot isolation transaction in an allocation unit.</td>
</tr>
<tr>
<td><code>min_record_size_in_bytes</code></td>
<td>Minimum record size in bytes.</td>
</tr>
<tr>
<td><code>max_record_size_in_bytes</code></td>
<td>Maximum record size in bytes.</td>
</tr>
<tr>
<td><code>avg_record_size_in_bytes</code></td>
<td>Average record size in bytes.</td>
</tr>
<tr>
<td><code>forwarded_record_count</code></td>
<td>Number of forwarded records in a heap.</td>
</tr>
</tbody>
</table>
Fixing

- Clustered and Non-clustered indexes
  - ALTER INDEX … REBUILD
    - Rebuild a new index, built side by side
    - Can run as a parallel operation
    - Offline operation except for Enterprise Edition
  - ALTER INDEX … REORGANIZE
    - First moves page data to the left side of the index to get the pages as full as possible and then removes any unneeded pages, then reorders the pages
    - Does not correct extent fragmentation
    - Can not run as a parallel operation
    - Online operation

- HEAP
  - Create a clustered index on the table and then drop the clustered index
  - Or create new table and move data to the new table
# Rebuild vs Reorganize

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Rebuild</th>
<th>Reorganize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online/Offline</td>
<td>Offline / (Online Enterprise)</td>
<td>Online</td>
</tr>
<tr>
<td>Faster when logical fragmentation is:</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Parallel processing</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Compacts pages</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can be stopped and restarted without losing work completed to that point</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Able to untangle interleaved indexes</td>
<td>May reduce interleaving</td>
<td>No</td>
</tr>
<tr>
<td>Additional free space is required in the data file for defragmenting</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Faster on larger indexes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Rebuilds statistics</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Log space usage</td>
<td>High in full recovery mode (logs entire contents of the index), low in bulk logged or simple recovery mode (only logs allocation of space)</td>
<td>Varies based on the amount of work performed</td>
</tr>
<tr>
<td>May skip pages on busy systems</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Rebuild or Reorganize

Guideline when to rebuild versus reorganize based on the `avg_fragmentation_in_percent` value

<table>
<thead>
<tr>
<th><code>avg_fragmentation_in_percent</code> value</th>
<th>Corrective statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 30%</td>
<td>ALTER INDEX REORGANIZE</td>
</tr>
<tr>
<td>&gt; 30%</td>
<td>ALTER INDEX REBUILD</td>
</tr>
</tbody>
</table>
ALTER INDEX … REBUILD

ALTER INDEX { index_name | ALL }
    ON <object>
    REBUILD
WITH
    PAD_INDEX = { ON | OFF }
    FILLFACTOR = fillfactor {1-100}
    SORT_IN_TEMPDB = { ON | OFF }
    IGNORE_DUP_KEY = { ON | OFF }
    STATISTICS_NORECOMPUTE = { ON | OFF }
    ONLINE = { ON | OFF }
    ALLOW_ROW_LOCKS = { ON | OFF }
    ALLOW_PAGE_LOCKS = { ON | OFF }
    MAXDOP = max_degree_of_parallelism { 0 = all or specify value }

Note: The underlined value is the default
ALTER INDEX ... REBUILD

- **FILLFACTOR** – how full to make the leaf level pages of the index. This is a percentage from 1-100, the default is 0 which is the same as 100.
- **PAD_INDEX** – specifies whether you want to leave free space in the intermediate pages. The fillfactor value is used for this, either the saved value or the value you specify in the command.
- **SORT_IN_TEMPDB** – this specifies whether to use the TempDB database to do a sort or to use the user database. If there is enough memory to sort the index this will all be done in memory instead.
- **IGNORE_DUP_KEY** – this tells SQL whether to continue or fail the index build if there is a duplicate key.
- **STATISTICS_NORECOMPUTE** – this tells SQL whether to re-compute the statistics for the index.
- **ONLINE** – this allows the index to be built online, so the there is no locking of the table or index. This is only available for the Enterprise Edition.
- **ALLOW_ROW_LOCKS** – tells SQL whether to use row locking when building the index
- **ALLOW_PAGE_LOCKS** - tells SQL whether to use page level locking when building the index
- **MAXDOP** – specifies how many processors to use for the index build. Only available in the Enterprise Edition.

ALTER INDEX … REBUILD

ALTER INDEX PK_Employee_EmployeeID ON HumanResources.Employee REBUILD;
ALTER INDEX ALL ON Production.Product REBUILD
WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON, STATISTICS_NORECOMPUTE = ON);
ALTER INDEX … REORGANIZE

ALTER INDEX { index_name | ALL }
ON <object>
REORGANIZE
WITH
  LOB_COMPACTION  = { ON | OFF }

Note: The underlined value is the default
ALTER INDEX … REORGANIZE

- **LOB_COMPACTION** - Specifies that all pages that contain large object (LOB) data are compacted. LOB data types are `image`, `text`, `ntext`, `varchar(max)`, `nvarchar(max)`, `varbinary(max)`, and `xml`. 
ALTER INDEX ... REORGANIZE

ALTER INDEX PK_ProductPhoto_ProductPhotoID
ON Production.ProductPhoto
REORGANIZE ;
Partitioned Indexes - Rebuild

ALTER INDEX \{ index_name | ALL \}
ON <object>
REBUILD
PARTITION = partition_number
WITH
SORT_IN_TEMPDB = \{ ON | OFF \}
MAXDOP = max_degree_of_parallelism
Partitioned Indexes - Reorganize

```
ALTER INDEX { index_name | ALL } 
  ON <object>
  REORGANIZE 
  PARTITION = partition_number 
WITH 
  LOB_COMPACTION = { ON | OFF }
```
Partitioned Indexes

ALTER INDEX IX_TransactionHistory
ON Production.TransactionHistory
REBUILD
Partition = 5;

ALTER INDEX IX_TransactionHistory
ON Production.TransactionHistory
REORGANIZE
Partition = 5;
DROP EXISTING

- You can use the DROP_EXISTING clause to rebuild the index, add or drop columns, modify options, modify column sort order, or change the partition scheme or filegroup.
- If the index enforces a PRIMARY KEY or UNIQUE constraint and the index definition is not altered in any way, the index is dropped and re-created preserving the existing constraint.
- DROP_EXISTING enhances performance when you re-create a clustered index, with either the same or different set of keys, on a table that also has nonclustered indexes.
- The nonclustered indexes are rebuilt once, and then only if the index definition has changed.
- The DROP_EXISTING clause does not rebuild the nonclustered indexes when the index definition has the same index name, key and partition columns, uniqueness attribute, and sort order as the original index.
DROP EXISTING

CREATE CLUSTERED INDEX
    IX_WorkOrder_ProductID
ON Production.WorkOrder(ProductID)
WITH
    (DROP_EXISTING = ON);
HEAP

CREATE CLUSTERED INDEX IX_WorkOrder ON Production.WorkOrder(ProductID)

DROP Production.WorkOrder.IX_WorkOrder
Managing

- Collecting data
- Selective rebuilds / reorgs
- Removing unused indexes
- Recovery Models and impact
- Transaction Log Usage
  - DBCC SQLPERF(logspace)
- Online rebuilds
- MAXDOP
- Changing index settings
  - PAD INDEX
  - FILLFACTOR
- Maintenance Plans
Impact by Recovery Model

<table>
<thead>
<tr>
<th>Index operation</th>
<th>Database Recovery Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full</td>
</tr>
<tr>
<td>ALTER INDEX REORGANIZE</td>
<td>Fully logged</td>
</tr>
<tr>
<td>ALTER INDEX REBUILD</td>
<td>Fully logged</td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>Fully logged</td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>Index page deallocation is fully logged; new heap rebuild, if applicable, is fully logged.</td>
</tr>
</tbody>
</table>

Monitoring Using Other Tools
Questions and Wrap-up

• Thanks to our sponsor: Idera
• Next webcast in the series:
  – Database Mirroring Concepts
  – May 13th, 2009, 4pm EDT