**Clustered Index Structures**

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 intermediate level 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.

Clustered indexes have one row in [sys.partitions](http://msdn.microsoft.com/en-us/library/ms175012(SQL.100).aspx), with **index\_id** = 1 for each partition used by the index. By default, a clustered index has a single partition. When a clustered index has multiple partitions, each partition has a B-tree structure that contains the data for that specific partition. For example, if a clustered index has four partitions, there are four B-tree structures; one in each partition.

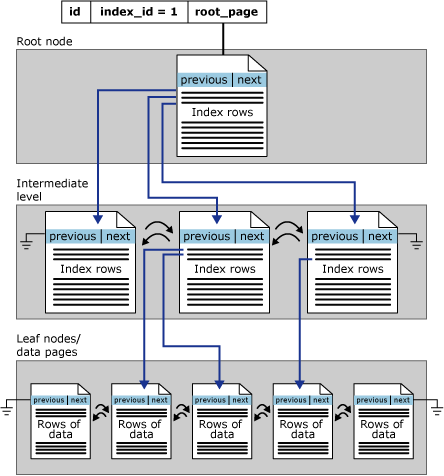
Depending on the data types in the clustered index, each clustered index structure will have one or more allocation units in which to store and manage the data for a specific partition. At a minimum, each clustered index will have one IN\_ROW\_DATA allocation unit per partition. The clustered index will also have one LOB\_DATA allocation unit per partition if it contains large object (LOB) columns. It will also have one ROW\_OVERFLOW\_DATA allocation unit per partition if it contains variable length columns that exceed the 8,060 byte row size limit. For more information about allocation units, see [Table and Index Organization](http://msdn.microsoft.com/en-us/library/ms189051(SQL.100).aspx).

The pages in the data chain and the rows in them are ordered on the value of the clustered index key. All inserts are made at the point where the key value in the inserted row fits in the ordering sequence among existing rows. The page collections for the B-tree are anchored by page pointers in the **sys.system\_internals\_allocation\_units** system view.

|  |
| --- |
| **Description: Important noteImportant** |
| The **sys.system\_internals\_allocation\_units** system view is reserved for MicrosoftSQL Server internal use only. Future compatibility is not guaranteed. |

For a clustered index, the **root\_page** column in **sys.system\_internals\_allocation\_units** points to the top of the clustered index for a specific partition. SQL Server moves down the index to find the row corresponding to a clustered index key. To find a range of keys, SQL Server moves through the index to find the starting key value in the range and then scans through the data pages using the previous or next pointers. To find the first page in the chain of data pages, SQL Server follows the leftmost pointers from the root node of the index.

This illustration shows the structure of a clustered index in a single partition.



**Heap Structures**

A heap is a table without a clustered index. Heaps have one row in [sys.partitions](http://msdn.microsoft.com/en-us/library/ms175012(SQL.100).aspx), with **index\_id** = 0 for each partition used by the heap. By default, a heap has a single partition. When a heap has multiple partitions, each partition has a heap structure that contains the data for that specific partition. For example, if a heap has four partitions, there are four heap structures; one in each partition.

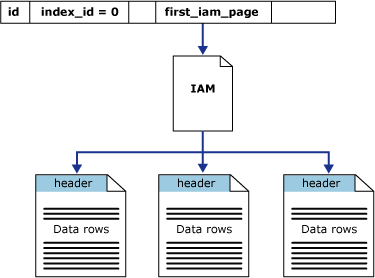
Depending on the data types in the heap, each heap structure will have one or more allocation units to store and manage the data for a specific partition. At a minimum, each heap will have one IN\_ROW\_DATA allocation unit per partition. The heap will also have one LOB\_DATA allocation unit per partition, if it contains large object (LOB) columns. It will also have one ROW\_OVERFLOW\_DATA allocation unit per partition, if it contains variable length columns that exceed the 8,060 byte row size limit. For more information about allocation units, see [Table and Index Organization](http://msdn.microsoft.com/en-us/library/ms189051(SQL.100).aspx).

The column **first\_iam\_page** in the **sys.system\_internals\_allocation\_units** system view points to the first IAM page in the chain of IAM pages that manage the space allocated to the heap in a specific partition. SQL Server uses the IAM pages to move through the heap. The data pages and the rows within them are not in any specific order and are not linked. The only logical connection between data pages is the information recorded in the IAM pages.

|  |
| --- |
| **Description: Important noteImportant** |
| The **sys.system\_internals\_allocation\_units** system view is reserved for MicrosoftSQL Server internal use only. Future compatibility is not guaranteed. |

Table scans or serial reads of a heap can be performed by scanning the IAM pages to find the extents that are holding pages for the heap. Because the IAM represents extents in the same order that they exist in the data files, this means that serial heap scans progress sequentially through each file. Using the IAM pages to set the scan sequence also means that rows from the heap are not typically returned in the order in which they were inserted.

The following illustration shows how the SQL Server Database Engine uses IAM pages to retrieve data rows in a single partition heap.



**Nonclustered Index Structures**

Nonclustered indexes have the same B-tree structure as clustered indexes, except for the following significant differences:

* The data rows of the underlying table are not sorted and stored in order based on their nonclustered keys.
* The leaf layer of a nonclustered index is made up of index pages instead of data pages.

Nonclustered indexes can be defined on a table or view with a clustered index or a heap. Each index row in the nonclustered index contains the nonclustered key value and a row locator. This locator points to the data row in the clustered index or heap having the key value.

The row locators in nonclustered index rows are either a pointer to a row or are a clustered index key for a row, as described in the following:

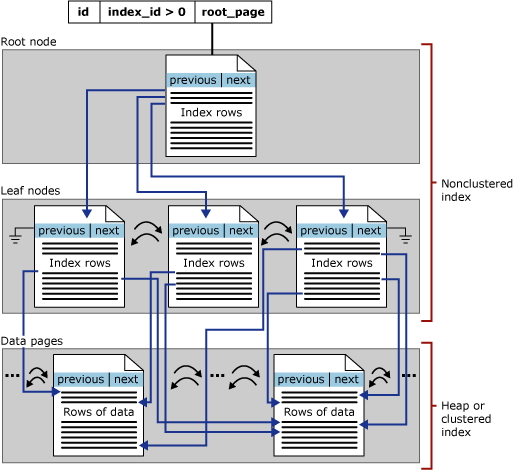
* If the table is a heap, which means it does not have a clustered index, the row locator is a pointer to the row. The pointer is built from the file identifier (ID), page number, and number of the row on the page. The whole pointer is known as a Row ID (RID).
* If the table has a clustered index, or the index is on an indexed view, the row locator is the clustered index key for the row. If the clustered index is not a unique index, SQL Server makes any duplicate keys unique by adding an internally generated value called a **uniqueifier**. This four-byte value is not visible to users. It is only added when required to make the clustered key unique for use in nonclustered indexes. SQL Server retrieves the data row by searching the clustered index using the clustered index key stored in the leaf row of the nonclustered index.

Nonclustered indexes have one row in [sys.partitions](http://msdn.microsoft.com/en-us/library/ms175012(SQL.100).aspx) with **index\_id** >0 for each partition used by the index. By default, a nonclustered index has a single partition. When a nonclustered index has multiple partitions, each partition has a B-tree structure that contains the index rows for that specific partition. For example, if a nonclustered index has four partitions, there are four B-tree structures, with one in each partition.

Depending on the data types in the nonclustered index, each nonclustered index structure will have one or more allocation units in which to store and manage the data for a specific partition. At a minimum, each nonclustered index will have one IN\_ROW\_DATA allocation unit per partition that stores the index B-tree pages. The nonclustered index will also have one LOB\_DATA allocation unit per partition if it contains large object (LOB) columns . Additionally, it will have one ROW\_OVERFLOW\_DATA allocation unit per partition if it contains variable length columns that exceed the 8,060 byte row size limit. For more information about allocation units, see [Table and Index Organization](http://msdn.microsoft.com/en-us/library/ms189051(SQL.100).aspx). The page collections for the B-tree are anchored by **root\_page** pointers in the **sys.system\_internals\_allocation\_units** system view.

|  |
| --- |
| **Description: Important noteImportant** |
| The **sys.system\_internals\_allocation\_units** system view is reserved for Microsoft SQL Server internal use only. Future compatibility is not guaranteed. |

The following illustration shows the structure of a nonclustered index in a single partition.



Description: http://i.msdn.microsoft.com/Global/Images/clear.gifIncluded Column Indexes

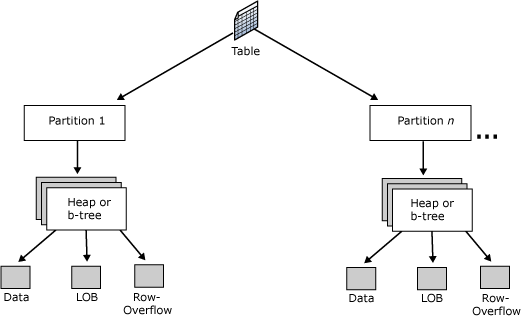
The functionality of nonclustered indexes can be extended by adding included columns, called nonkey columns, to the leaf level of the index. While the key columns are stored at all levels of the nonclustered index, nonkey columns are stored only at the leaf level. For more information, see [Index with Included Columns](http://msdn.microsoft.com/en-us/library/ms190806(SQL.100).aspx).

**Table and Index Organization**

Tables and indexes are stored as a collection of 8-KB pages. This topic describes the way table and index pages are organized.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifTable Organization

The following illustration shows the organization of a table. A table is contained in one or more partitions and each partition contains data rows in either a heap or a clustered index structure. The pages of the heap or clustered index are managed in one or more allocation units, depending on the column types in the data rows.



Description: http://i.msdn.microsoft.com/Global/Images/clear.gifPartitions

Table and index pages are contained in one or more partitions. A partition is a user-defined unit of data organization. By default, a table or index has only one partition that contains all the table or index pages. The partition resides in a single filegroup. A table or index with a single partition is equivalent to the organizational structure of tables and indexes in earlier versions of SQL Server.

When a table or index uses multiple partitions, the data is partitioned horizontally so that groups of rows are mapped into individual partitions, based on a specified column. The partitions can be put on one or more filegroups in the database. The table or index is treated as a single logical entity when queries or updates are performed on the data. For more information, see [Partitioned Tables and Indexes](http://msdn.microsoft.com/en-us/library/ms188706(SQL.100).aspx).

To view the partitions used by a table or index, use the [sys.partitions (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms175012(SQL.100).aspx) catalog view.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifClustered Tables, Heaps, and Indexes

SQL Server tables use one of two methods to organize their data pages within a partition:

* Clustered tables are tables that have a clustered index.

The data rows are stored in order based on the clustered index key. The clustered index is implemented as a B-tree index structure that supports fast retrieval of the rows, based on their clustered index key values. The pages in each level of the index, including the data pages in the leaf level, are linked in a doubly-linked list. However, navigation from one level to another is performed by using key values. For more information, see [Clustered Index Structures](http://msdn.microsoft.com/en-us/library/ms177443(SQL.100).aspx).

* Heaps are tables that have no clustered index.

The data rows are not stored in any particular order, and there is no particular order to the sequence of the data pages. The data pages are not linked in a linked list. For more information, see [Heap Structures](http://msdn.microsoft.com/en-us/library/ms188270(SQL.100).aspx).

Indexed views have the same storage structure as clustered tables.

When a heap or a clustered table has multiple partitions, each partition has a heap or B-tree structure that contains the group of rows for that specific partition. For example, if a clustered table has four partitions, there are four B-trees; one in each partition.

**Nonclustered Indexes**

Nonclustered indexes have a B-tree index structure similar to the one in clustered indexes. The difference is that nonclustered indexes do not affect the order of the data rows. The leaf level contains index rows. Each index row contains the nonclustered key value, a row locator and any included, or nonkey, columns. The locator points to the data row that has the key value. For more information, see [Nonclustered Index Structures](http://msdn.microsoft.com/en-us/library/ms177484(SQL.100).aspx).

**XML Indexes**

One primary and several secondary XML indexes can be created on each **xml** column in the table. An XML index is a shredded and persisted representation of the XML binary large objects (BLOBs) in the **xml** data type column. XML indexes are stored as internal tables. To view information about xml indexes, use the [sys.xml\_indexes](http://msdn.microsoft.com/en-us/library/ms176003(SQL.100).aspx) or [sys.internal\_tables](http://msdn.microsoft.com/en-us/library/ms187799(SQL.100).aspx) catalog views.

For more information about XML indexes, see [Indexes on XML Data Type Columns](http://msdn.microsoft.com/en-us/library/ms191497(SQL.100).aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAllocation Units

An allocation unit is a collection of pages within a heap or B-tree used to manage data based on their page type. The following table lists the types of allocation units used to manage data in tables and indexes.

|  |  |
| --- | --- |
| **Allocation unit type** | **Is used to manage** |
| IN\_ROW\_DATA | Data or index rows that contain all data, except large object (LOB) data.  Pages are of type Data or Index. |
| LOB\_DATA | 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 | 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 Text/Image. |

For more information about page types, see [Understanding Pages and Extents](http://msdn.microsoft.com/en-us/library/ms190969(SQL.100).aspx).

A heap or B-tree can have only one allocation unit of each type in a specific partition. To view the table or index allocation unit information, use the [sys.allocation\_units](http://msdn.microsoft.com/en-us/library/ms189792(SQL.100).aspx) catalog view.

**IN\_ROW\_DATA Allocation Unit**

For every partition used by a table (heap or clustered table), index, or indexed view, there is one IN\_ROW\_DATA allocation unit that is made up of a collection of data pages. This allocation unit also contains additional collections of pages to implement each nonclustered and XML index defined for the table or view. The page collections in each partition of a table, index, or indexed view are anchored by page pointers in the **sys.system\_internals\_allocation\_units** system view.

|  |
| --- |
| **Description: Important noteImportant** |
| The **sys.system\_internals\_allocation\_units** system view is reserved for MicrosoftSQL Server internal use only. Future compatibility not guaranteed. |

Each table, index, and indexed view partition has a row in **sys.system\_internals\_allocation\_units** uniquely identified by a container ID (**container\_id**). The container ID has a one-to-one mapping to the **partition\_id** in the **sys.partitions** catalog view that maintains the relationship between the table, index, or the indexed view data stored in a partition and the allocation units used to manage the data within the partition.

The allocation of pages to a table, index, or an indexed view partition is managed by a chain of IAM pages. The column **first\_iam\_page** in **sys.system\_internals\_allocation\_units** points to the first IAM page in the chain of IAM pages managing the space allocated to the table, index, or the indexed view in the IN\_ROW\_DATA allocation unit.

**sys.partitions** returns a row for each partition in a table or index.

* A heap has a row in **sys.partitions** with **index\_id** = 0.

The **first\_iam\_page** column in **sys.system\_internals\_allocation\_units** points to the IAM chain for the collection of heap data pages in the specified partition. The server uses the IAM pages to find the pages in the data page collection, because they are not linked.

* A clustered index on a table or a view has a row in **sys.partitions** with **index\_id** = 1.

The **root\_page** column in **sys.system\_internals\_allocation\_units** points to the top of the clustered index B-tree in the specified partition. The server uses the index B-tree to find the data pages in the partition.

* Each nonclustered index created for a table or a view has a row in **sys.partitions** with **index\_id** > 1.

The **root\_page** column in **sys.system\_internals\_allocation\_units** points to the top of the nonclustered index B-tree in the specified partition.

* Each table that has at least one LOB column also has a row in **sys.partitions** with **index\_id** > 250.

The **first\_iam\_page** column points to the chain of IAM pages that manage the pages in the LOB\_DATA allocation unit.

**ROW\_OVERFLOW\_DATA Allocation Unit**

For every partition used by a table (heap or clustered table), index, or indexed view, there is one ROW\_OVERFLOW\_DATA allocation unit. This allocation unit contains zero (0) pages until a data row with variable length columns (**varchar**, **nvarchar**, **varbinary**, or **sql\_variant**) in the IN\_ROW\_DATA allocation unit exceeds the 8 KB row size limit. When the size limitation is reached, SQL Server moves the column with the largest width from that row to a page in the ROW\_OVERFLOW\_DATA allocation unit. A 24-byte pointer to this off-row data is maintained on the original page.

Text/Image pages in the ROW\_OVERFLOW\_DATA allocation unit are managed in the same way pages in the LOB\_DATA allocation unit are managed. That is, the Text/Image pages are managed by a chain of IAM pages.

**LOB\_DATA Allocation Unit**

When a table or index has one or more LOB data types, one LOB\_DATA allocation unit per partition is allocated to manage the storage of that data. The LOB data types include **text**, **ntext**, **image**, **xml**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, and CLR user-defined types.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifPartition and Allocation Unit Example

The following example returns partition and allocation unit data for two tables: DatabaseLog, a heap with LOB data and no nonclustered indexes, and Currency, a clustered table without LOB data and one nonclustered index. Both tables have a single partition.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl28_ctl00_ctl00_code');" \o "Copy Code)

USE AdventureWorks;

GO

SELECT o.name AS table\_name,p.index\_id, i.name AS index\_name , au.type\_desc AS allocation\_type, au.data\_pages, partition\_number

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

WHERE o.name = N'DatabaseLog' OR o.name = N'Currency'

ORDER BY o.name, p.index\_id;

Here is the result set. Notice that the DatabaseLog table uses all three allocation unit types, because it contains both data and Text/Image page types. The Currency table does not have LOB data, but does have the allocation unit required to manage data pages. If the Currency table is later modified to include a LOB data type column, a LOB\_DATA allocation unit is created to manage that data.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl28_ctl00_ctl01_code');" \o "Copy Code)

table\_name index\_id index\_name allocation\_type data\_pages partition\_number

----------- -------- ----------------------- --------------- ----------- ------------

Currency 1 PK\_Currency\_CurrencyCode IN\_ROW\_DATA 1 1

Currency 3 AK\_Currency\_Name IN\_ROW\_DATA 1 1

DatabaseLog 0 NULL IN\_ROW\_DATA 160 1

DatabaseLog 0 NULL ROW\_OVERFLOW\_DATA 0 1

DatabaseLog 0 NULL LOB\_DATA 49 1

(5 row(s) affected)