Pages and Extents Architecture Guide

- 03/12/2019
- 18 minutes to read
-

Applies to: ✓ SQL Server (all supported versions) ✓ Azure SQL Database ✓ Azure SQL Managed Instance ✓ Azure Synapse Analytics ✓ Parallel Data Warehouse

The page is the fundamental unit of data storage in SQL Server. An extent is a collection of eight physically contiguous pages. Extents help efficiently manage pages. This guide describes the data structures that are used to manage pages and extents in all versions of SQL Server. Understanding the architecture of pages and extents is important for designing and developing databases that perform efficiently.

Pages and Extents

The fundamental unit of data storage in SQL Server is the page. The disk space allocated to a data file (.mdf or .ndf) in a database is logically divided into pages numbered contiguously from 0 to n. Disk I/O operations are performed at the page level. That is, SQL Server reads or writes whole data pages.

Extents are a collection of eight physically contiguous pages and are used to efficiently manage the pages. All pages are organized into extents.

Pages

Take a regular book: all content in it is written on pages. Similar to a book, in SQL Server all the data rows are written on pages. In a book, all pages are the same physical size. Similarly, in SQL Server all data pages are the same size - 8 kilobytes. In a book most pages contain the data - the main content of the book - and some pages contain metadata about the content - for example table of contents and index. Again, SQL Server is not different: most pages contain actual rows of data which were stored by users; these are called Data pages and text/image pages (for special cases). The Index pages contain index references about where the data is and finally there are system pages that store variety of metadata about the organization of the data (PFS, GAM, SGAM, IAM, DCM, BCM pages). See table below for page types and their description.

As mentioned, in SQL Server, the page size is 8-KB. This means SQL Server databases have 128 pages per megabyte. Each page begins with a 96-byte header that is used to store system information about the page. This information includes the page number, page type, the amount of free space on the page, and the allocation unit ID of the object that owns the page.

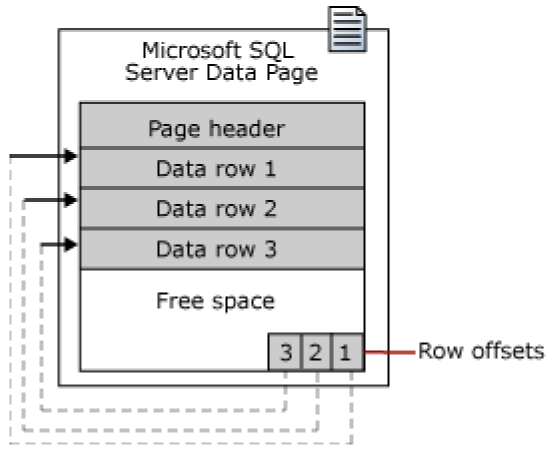
The following table shows the page types used in the data files of a SQL Server database.

Page type	Contents
Data	Data rows with all data, except text, ntext, image, nvarchar(max), varchar(max), varbinary(max), and xml data, when text in row is set to ON.
Index	Index entries.
Text/Image	Large object data types: (text, ntext, image, nvarchar(max), varchar(max), varbinary(max), and xml data) Variable length columns when the data row exceeds 8 KB: (varchar, nvarchar, varbinary, and sql_variant)
Global Allocation Map, Shared Global Allocation Map	Information about whether extents are allocated.
Page Free Space (PFS)	Information about page allocation and free space available on pages.
Index Allocation Map	Information about extents used by a table or index per allocation unit.
Bulk Changed Map	Information about extents modified by bulk operations since the last BACKUP LOG statement per allocation unit.
Differential Changed Map	Information about extents that have changed since the last BACKUP DATABASE statement per allocation unit.

Note

Log files do not contain pages; they contain a series of log records.

Data rows are put on the page serially, starting immediately after the header. A row offset table starts at the end of the page, and each row offset table contains one entry for each row on the page. Each row offset entry records how far the first byte of the row is from the start of the page. Thus, the function of the row offset table is to help SQL Server locate rows on a page very quickly. The entries in the row offset table are in reverse sequence from the sequence of the rows on the page.



Large Row Support

Rows cannot span pages, however portions of the row may be moved off the row's page so that the row can actually be very large. The maximum amount of data and overhead that is contained in a single row on a page is 8,060 bytes (8-KB). However, this does not include the data stored in the Text/Image page type.

This restriction is relaxed for tables that contain varchar, nvarchar, varbinary, or sql_variant columns. When the total row size of all fixed and variable columns in a table exceeds the 8,060-byte limitation, SQL Server dynamically moves one or more variable length columns to pages in the ROW_OVERFLOW_DATA allocation unit, starting with the column with the largest width.

This is done whenever an insert or update operation increases the total size of the row beyond the 8,060-byte limit. When a column is moved to a page in the ROW_OVERFLOW_DATA allocation unit, a 24-byte pointer on the original page in the IN_ROW_DATA allocation unit is maintained. If a subsequent operation reduces the row size, SQL Server dynamically moves the columns back to the original data page.

Row-Overflow Considerations

As mentioned earlier, a row cannot reside on multiple pages and can overflow if the combined size of variable-length data-type fields exceeds the 8060-byte limit. To illustrate, a table may be created with two columns: one varchar(7000) and another varchar (2000). Individually neither column exceeds the 8060-byte, but combined they could do so, if the entire width of each column is filled. SQL Server may dynamically move the varchar(7000) variable length column to pages in the ROW_OVERFLOW_DATA allocation unit. When you combine varchar, nvarchar, varbinary, sql_variant, or CLR user-defined type columns that exceed 8,060 bytes per row, consider the following:

- Moving large records to another page occurs dynamically as records are lengthened based on update operations. Update operations that shorten records may cause records to be moved back to the original page in the

IN_ROW_DATA allocation unit. Querying and performing other select operations, such as sorts or joins on large records that contain row-overflow data slows processing time, because these records are processed synchronously instead of asynchronously.
Therefore, when you design a table with multiple varchar, nvarchar, varbinary, sql_variant, or CLR user-defined type columns, consider the percentage of rows that are likely to flow over and the frequency with which this overflow data is likely to be queried. If there are likely to be frequent queries on many rows of row-overflow data, consider normalizing the table so that some columns are moved to another table. This can then be queried in an asynchronous JOIN operation.

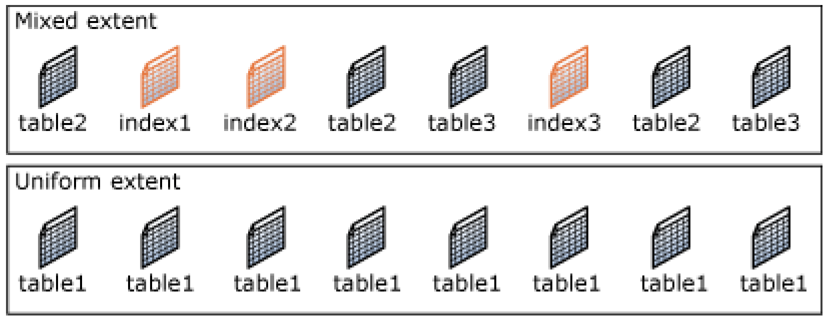
- The length of individual columns must still fall within the limit of 8,000 bytes for varchar, nvarchar, varbinary, sql_variant, and CLR user-defined type columns. Only their combined lengths can exceed the 8,060-byte row limit of a table.
- The sum of other data type columns, including char and nchar data, must fall within the 8,060-byte row limit. Large object data is also exempt from the 8,060-byte row limit.
- The index key of a clustered index cannot contain varchar columns that have existing data in the ROW_OVERFLOW_DATA allocation unit. If a clustered index is created on a varchar column and the existing data is in the IN_ROW_DATA allocation unit, subsequent insert or update actions on the column that would push the data off-row will fail. For more information about allocation units, see Table and Index Organization.
- You can include columns that contain row-overflow data as key or nonkey columns of a nonclustered index.
- The record-size limit for tables that use sparse columns is 8,018 bytes. When the converted data plus existing record data exceeds 8,018 bytes, MSSQLSERVER ERROR 576 is returned. When columns are converted between sparse and nonsparse types, Database Engine keeps a copy of the current record data. This temporarily doubles the storage that is required for the record.
- To obtain information about tables or indexes that might contain row-overflow data, use the sys.dm_db_index_physical_stats dynamic management function.

Extents

Extents are the basic unit in which space is managed. An extent is eight physically contiguous pages, or 64 KB. This means SQL Server databases have 16 extents per megabyte.

SQL Server has two types of extents:

- **Uniform** extents are owned by a single object; all eight pages in the extent can only be used by the owning object.
- **Mixed** extents are shared by up to eight objects. Each of the eight pages in the extent can be owned by a different object.



Up to, and including, SQL Server 2014 (12.x), SQL Server does not allocate whole extents to tables with small amounts of data. A new table or index generally allocates pages from mixed extents. When the table or index grows to the point that it has eight pages, it then switches to use uniform extents for subsequent allocations. If you create an index on an existing table that has enough rows to generate eight pages in the index, all allocations to the index are in uniform extents.

Starting with SQL Server 2016 (13.x), the default for most allocations in a user database and tempdb is to use uniform extents, except for allocations belonging to the first eight pages of an IAM chain. Allocations for master, msdb, and model databases still retain the previous behavior.

Note

Up to, and including, SQL Server 2014 (12.x), trace flag 1118 can be used to change the default allocation to always use uniform extents. For more information about this trace flag, see **DBCC TRACEON - Trace Flags**.

Starting with SQL Server 2016 (13.x), the functionality provided by TF 1118 is automatically enabled for tempdb and all user databases. For user databases, this behavior is controlled by the SET MIXED_PAGE_ALLOCATION option of ALTER DATABASE, with the default value set to OFF, and trace flag 1118 has no effect. For more information, see **ALTER DATABASE SET Options (Transact-SQL)**.

Starting with SQL Server 2012 (11.x), the sys.dm_db_database_page_allocations system function can report page allocation information for a database, table, index, and partition.

Important

The sys.dm_db_database_page_allocations system function is not documented and is subject to change. Compatibility is not guaranteed.

Starting with SQL Server 2019 (15.x), the sys.dm_db_page_info system function is available and returns information about a page in a database. The function returns one row that contains the header information from the page, including the object_id, index_id, and partition_id. This function replaces the need to use DBCC PAGE in most cases.

Managing Extent Allocations and Free Space

The SQL Server data structures that manage extent allocations and track free space have a relatively simple structure. This has the following benefits:

- The free space information is densely packed, so relatively few pages contain this information.
  This increases speed by reducing the amount of disk reads that are required to retrieve allocation information. This also increases the chance that the allocation pages will remain in memory and not require more reads.
- Most of the allocation information is not chained together. This simplifies the maintenance of the allocation information.
  Each page allocation or deallocation can be performed quickly. This decreases the contention between concurrent tasks having to allocate or deallocate pages.

Managing Extent Allocations

SQL Server uses two types of allocation maps to record the allocation of extents:

- **Global Allocation Map (GAM)**
  GAM pages record what extents have been allocated. Each GAM covers 64,000 extents, or almost 4 gigabytes (GB) of data. The GAM has 1-bit for each extent in the interval it covers. If the bit is 1, the extent is free; if the bit is 0, the extent is allocated.
- **Shared Global Allocation Map (SGAM)**
  SGAM pages record which extents are currently being used as mixed extents and also have at least one unused page. Each SGAM covers 64,000 extents, or almost 4-GB of data. The SGAM has 1-bit for each extent in the interval it covers. If the bit is 1, the extent is being used as a mixed extent and has a free page. If the bit is 0, the extent is not used as a mixed extent, or it is a mixed extent and all its pages are being used.

Each extent has the following bit patterns set in the GAM and SGAM, based on its current use.

MANAGING EXTENT ALLOCATIONS

Current use of extent	GAM bit setting	SGAM bit setting
Free, not being used	1	0
Uniform extent, or full mixed extent	0	0
Mixed extent with free pages	0	1

This causes simple extent management algorithms.

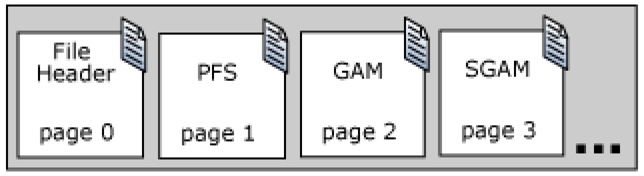
- To allocate a uniform extent, the SQL Server Database Engine searches the GAM for a 1 bit and sets it to 0.
- To find a mixed extent with free pages, the SQL Server Database Engine searches the SGAM for a 1 bit.
- To allocate a mixed extent, the SQL Server Database Engine searches the GAM for a 1 bit, sets it to 0, and then also sets the corresponding bit in the SGAM to 1.
- To deallocate an extent, the SQL Server Database Engine makes sure that the GAM bit is set to 1 and the SGAM bit is set to 0. The algorithms that are actually used internally by the SQL Server Database Engine are more sophisticated than what is described in this article, because the SQL Server Database Engine distributes data evenly in a database. However, even the real algorithms are simplified by not having to manage chains of extent allocation information.

Tracking free space

Page Free Space (PFS) pages record the allocation status of each page, whether an individual page has been allocated, and the amount of free space on each page. The PFS has 1-byte for each page, recording whether the page is allocated, and if so, whether it is empty, 1 to 50 percent full, 51 to 80 percent full, 81 to 95 percent full, or 96 to 100 percent full.

After an extent has been allocated to an object, the SQL Server Database Engine uses the PFS pages to record which pages in the extent are allocated or free. This information is used when the SQL Server Database Engine has to allocate a new page. The amount of free space in a page is only maintained for heap and Text/Image pages. It is used when the SQL Server Database Engine has to find a page with free space available to hold a newly inserted row. Indexes do not require that the page free space be tracked, because the point at which to insert a new row is set by the index key values.

A new PFS, GAM or SGAM page is added in the data file for every additional range that it keeps track of. Thus, there is a new PFS page 8,088 pages after the first PFS page, and additional PFS pages in subsequent 8,088 page intervals. To illustrate, page ID 1 is a PFS page, page ID 8088 is a PFS page, page ID 16176 is a PFS page, and so on. There is a new GAM page 64,000 extents after the first GAM page and it keeps track of the 64,000-extents following it; the sequence continues at 64,000-extent intervals. Similarly, there is a new SGAM page 64,000 extents after the first SGAM page and additional SGAM pages in subsequent 64,000 extent intervals. The following illustration shows the sequence of pages used by the SQL Server Database Engine to allocate and manage extents.



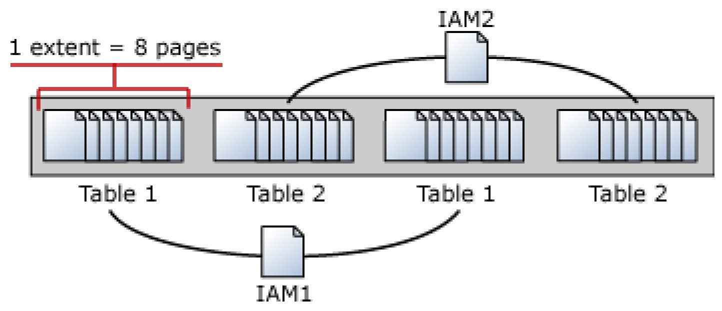
Managing space used by objects

An **Index Allocation Map (IAM)** page maps the extents in a 4-GB part of a database file used by an allocation unit. An allocation unit is one of three types:

- IN_ROW_DATA
  Holds a partition of a heap or index.
- LOB_DATA
  Holds large object (LOB) data types, such as XML, VARBINARY(max), and VARCHAR(max).
- ROW_OVERFLOW_DATA
  Holds variable length data stored in VARCHAR, NVARCHAR, VARBINARY, or SQL_VARIANT columns that exceed the 8,060 byte row size limit.

Each partition of a heap or index contains at least an IN_ROW_DATA allocation unit. It may also contain a LOB_DATA or ROW_OVERFLOW_DATA allocation unit, depending on the heap or index schema.

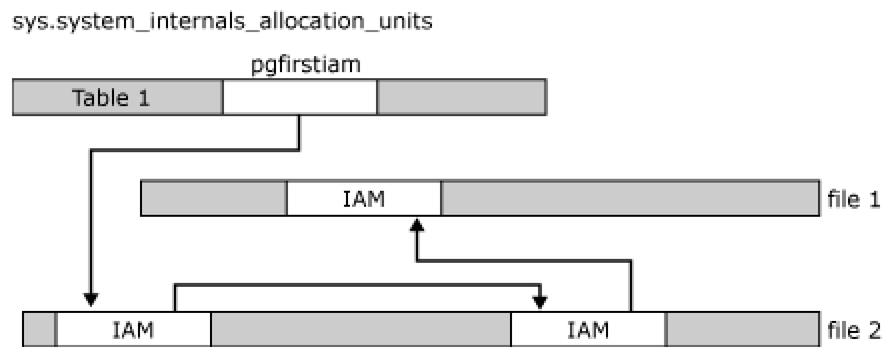
An IAM page covers a 4-GB range in a file and is the same coverage as a GAM or SGAM page. If the allocation unit contains extents from more than one file, or more than one 4-GB range of a file, there will be multiple IAM pages linked in an IAM chain. Therefore, each allocation unit has at least one IAM page for each file on which it has extents. There may also be more than one IAM page on a file, if the range of the extents on the file allocated to the allocation unit exceeds the range that a single IAM page can record.



IAM pages are allocated as required for each allocation unit and are located randomly in the file.
The sys.system_internals_allocation_units system view points to the first IAM page for an allocation unit. All the IAM pages for that allocation unit are linked in an IAM chain.

Important

The sys.system_internals_allocation_units system view is for internal use only and is subject to change. Compatibility is not guaranteed. This view is not available in Azure SQL Database.



IAM pages linked in a chain per allocation unit An IAM page has a header that indicates the starting extent of the range of extents mapped by the IAM page. The IAM page also has a large bitmap in which each bit represents one extent. The first bit in the map represents the first extent in the range, the second bit represents the second extent, and so on. If a bit is 0, the extent it represents is not allocated to the allocation unit owning the IAM. If the bit is 1, the extent it represents is allocated to the allocation unit owning the IAM page.

When the SQL Server Database Engine has to insert a new row and no space is available in the current page, it uses the IAM and PFS pages to find a page to allocate, or, for a heap or a Text/Image page, a page with sufficient space to hold the row. The SQL Server Database Engine uses the IAM pages to find the extents allocated to the allocation unit. For each extent, the SQL Server Database Engine searches the PFS pages to see if there is a page that can be used. Each IAM and PFS page covers lots of data pages, so there are few IAM and PFS pages in a database. This means that the IAM and PFS pages are generally in memory in the SQL Server buffer pool, so they can be searched quickly. For indexes, the insertion point of a new row is set by the index key, but when a new page is needed, the previously described process occurs.

The SQL Server Database Engine allocates a new extent to an allocation unit only when it cannot quickly find a page in an existing extent with sufficient space to hold the row being inserted.

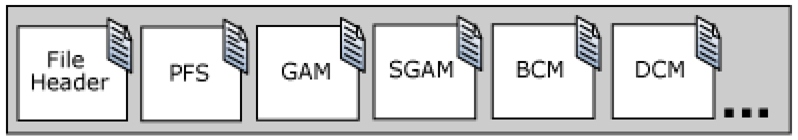
The SQL Server Database Engine allocates extents from those available in the filegroup using a **proportional fill allocation algorithm**. If in the same filegroup with two files, one file has two times the free space as the other, two pages will be allocated from the file with the available space for every one page allocated from the other file. This means that every file in a filegroup should have a similar percentage of space used.

Tracking Modified Extents

SQL Server uses two internal data structures to track extents modified by bulk copy operations and extents modified since the last full backup. These data structures greatly speed up differential backups. They also speed up the logging of bulk copy operations when a database is using the bulk-logged recovery model. Like the Global Allocation Map (GAM) and Shared Global Allocation Map (SGAM) pages, these structures are bitmaps in which each bit represents a single extent.

- **Differential Changed Map (DCM)**
  This tracks the extents that have changed since the last BACKUP DATABASE statement. If the bit for an extent is 1, the extent has been modified since the last BACKUP DATABASE statement. If the bit is 0, the extent has not been modified. Differential backups read just the DCM pages to determine which extents have been modified. This greatly reduces the number of pages that a differential backup must scan. The length of time that a differential backup runs is proportional to the number of extents modified since the last BACKUP DATABASE statement and not the overall size of the database.
- **Bulk Changed Map (BCM)**
  This tracks the extents that have been modified by bulk logged operations since the last BACKUP LOG statement. If the bit for an extent is 1, the extent has been modified by a bulk logged operation after the last BACKUP LOG statement. If the bit is 0, the extent has not been modified by bulk logged operations. Although BCM pages appear in all databases, they are only relevant when the database is using the bulk-logged recovery model. In this recovery model, when a BACKUP LOG is performed, the backup process scans the BCMs for extents that have been modified. It then includes those extents in the log backup. This lets the bulk logged operations be recovered if the database is restored from a database backup and a sequence of transaction log backups. BCM pages are not relevant in a database that is using the simple recovery model, because no bulk logged operations are logged. They are not relevant in a database that is using the full recovery model, because that recovery model treats bulk logged operations as fully logged operations.

The interval between DCM pages and BCM pages is the same as the interval between GAM and SGAM page, 64,000 extents. The DCM and BCM pages are located behind the GAM and SGAM pages in a physical file:


File Header
PFS
GAM
SGAM
BCM
DCM
...