Physical Organization: SQL Server 2005

Tables

- Tables and indexes are stored as a collection of 8 KB pages
- A table is divided in one or more **partitions**
- Each partition contains data rows in either a **heap** or a **clustered table**.
- 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.

Tables

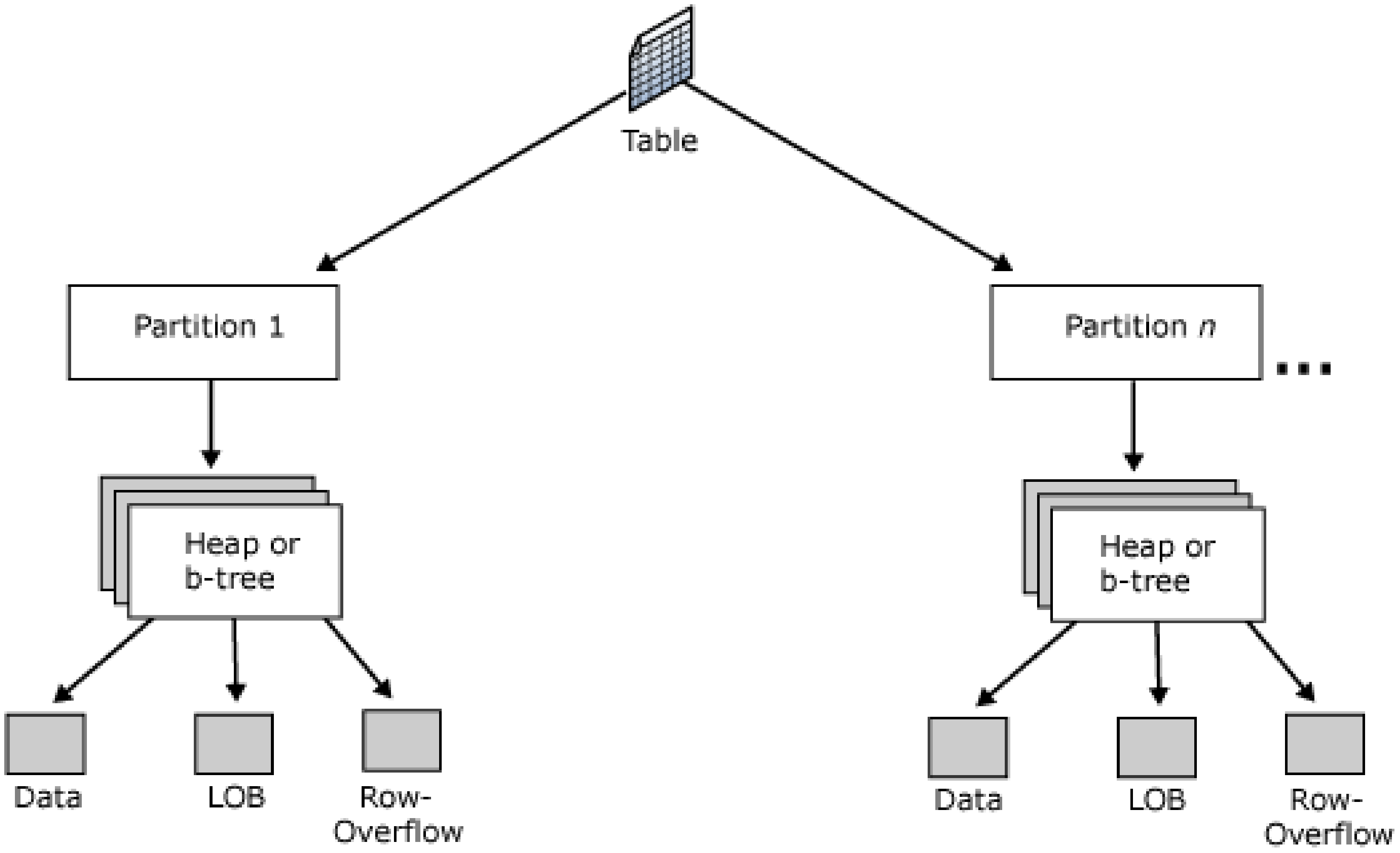


Table
Partition 1
Partition n
...
Heap or b-tree
Heap or b-tree
Data
LOB
Row-Overflow
Data
LOB
Row-Overflow

Partitions

- Table and index pages are divided in one or more partitions.
- 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.
- 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.

Partitions

- 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.
- To view the partitions used by a table or index, use the **sys.partitions** catalog view.

Organization of a Partition

- SQL Server 2005 tables use one of two methods to organize their data pages within a partition:
- **Clustered tables**: 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.
  - The pages in each level of the index, including the data pages in the leaf level, are linked in a doubly-linked list.
- **Heaps**: 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.

Allocation Units

- An allocation unit is a collection of pages within a heap or B+tree used to manage data based on their page type.

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 Data.

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.

ROW_OVERFLOW_DATA Allocation Unit

- For every partition used by a table (heap or clustered table), index, or indexed view, there can be 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.

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.

Heaps

- A heap stores a table without a clustered index.
- When a heap has multiple partitions, each partition has a heap structure that contains the data for that specific partition.
- At a minimum, each heap will have one IN_ROW_DATA allocation unit per partition. The heap may also have one LOB_DATA allocation unit per partition and one ROW_OVERFLOW_DATA allocation unit per partition.

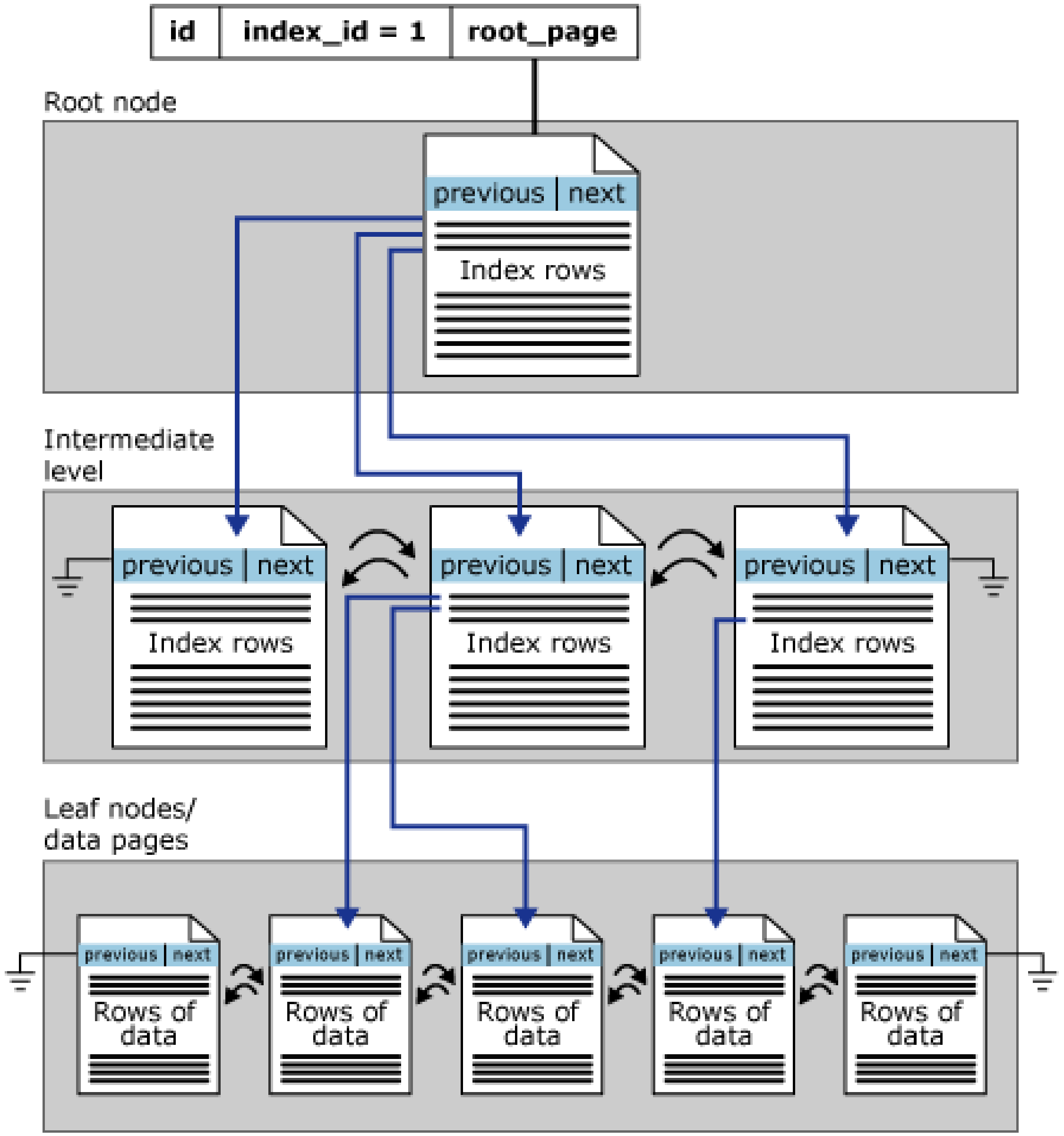
Clustered Tables

- They are organized as B+-trees
- One page per node
- The leaf nodes contain the data pages of the underlying table.
- Thus the data is stored inside the clustered index
- The pages in each level of the index are linked in a doubly-linked list.
- When a clustered index has multiple partitions, each partition has a B+-tree structure that contains the data for that specific partition.

Clustered Tables

- 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.
- At a minimum, each clustered index will have one IN_ROW_DATA allocation unit per partition. The clustered index may also have one LOB_DATA allocation unit per partition and one ROW_OVERFLOW_DATA allocation unit per partition.

A Clustered Table in a Single Partition



Nonclustered Indexes

- Same B+-tree structure as clustered tables but
  - The leaf layer of a nonclustered index is made up of index pages instead of data pages.
  - The data rows of the underlying table are not sorted and stored in order based on their nonclustered keys
- When a nonclustered index has multiple partitions, each partition has a B+-tree structure that contains the data for that specific partition.

Nonclustered Indexes

- Nonclustered indexes can be defined on a table or view with a clustered index or a heap.
- Each index row in the leaves of the nonclustered index contains the key value and a row locator. This locator points to the data row in the clustered index or heap having the key value.
- At a minimum, each nonclustered index will have one IN_ROW_DATA allocation unit per partition. The unclustered index may also have one LOB_DATA allocation unit per partition and one ROW_OVERFLOW_DATA allocation unit per partition.

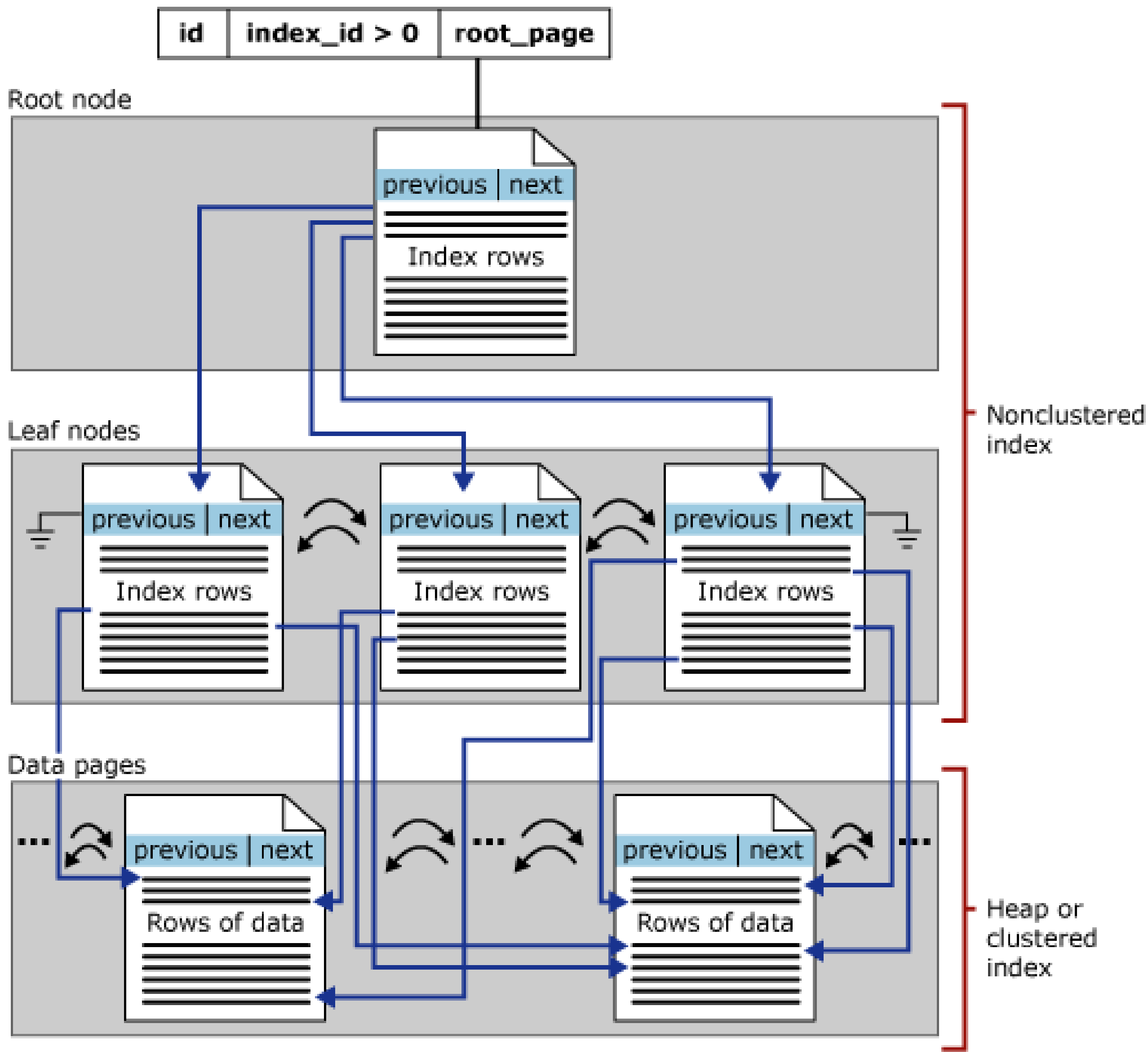
Row Locators

- If the table is a heap 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, the row locator is the clustered index key for the row.
  - If the clustered index is not a unique index, SQL Server 2005 makes any duplicate keys unique by adding an internally generated value called a **uniqueifier**. This four-byte value is not visible to users.
  - 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

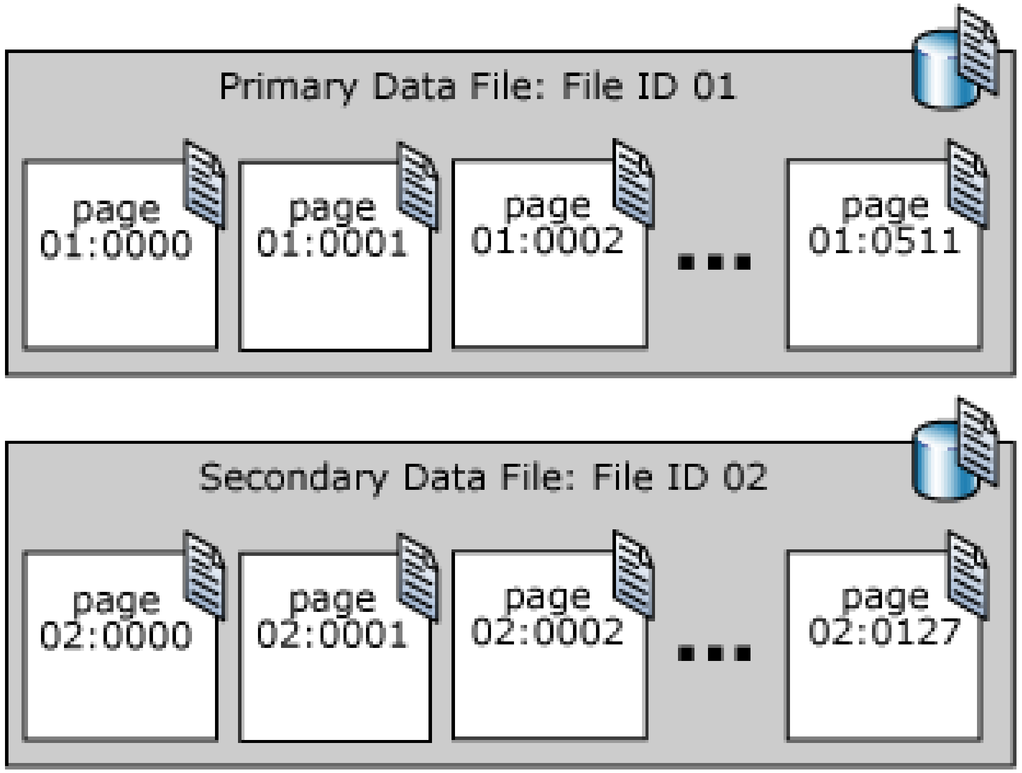
- When a nonclustered index has multiple partitions, each partition has a B+tree structure that contains the index rows for that specific partition.

A Nonclustered Index in a Single Partition



Pages

•Pages in file are numbered sequentially, starting from 0



Pages

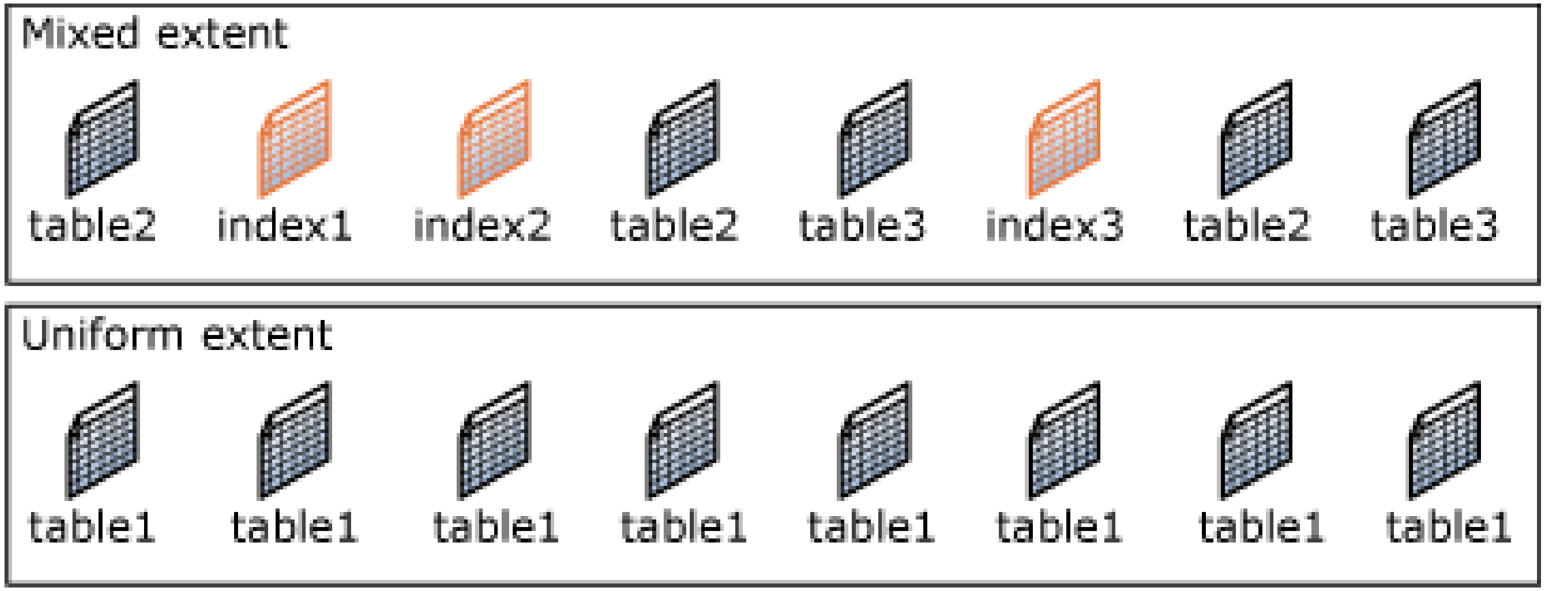
- The first page of a file contains information about the attributes of the file
- Other pages at the beginning of the file can be used for containing system information, such as allocation maps

Management of Space on Disks

- Space on disks is managed in **extents**
- 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.

Extents

- A new table or index is generally allocated 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.



Space Allocation

- The data structures that manage extent allocations and track free space have a relatively simple structure. Benefits:
  - The free space information is densely packed, so relatively few pages contain this information.
  - Most of the allocation information is not chained together. This simplifies the maintenance of the allocation information.

Space Allocation

- 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 GB of data. The GAM has one bit for each extent. 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 one bit for each extent. 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.

GAM and SGAM Bits

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

Space Allocation Algorithm

- To allocate a uniform extent, the Database Engine searches the GAM for a 1 bit and sets it to 0.
- To find a mixed extent with free pages, the Database Engine searches the SGAM for a 1 bit.
- To allocate a mixed extent, the 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 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 Database Engine are more sophisticated but they do not have to manage chains of extent allocation information.

Tracking Free Space

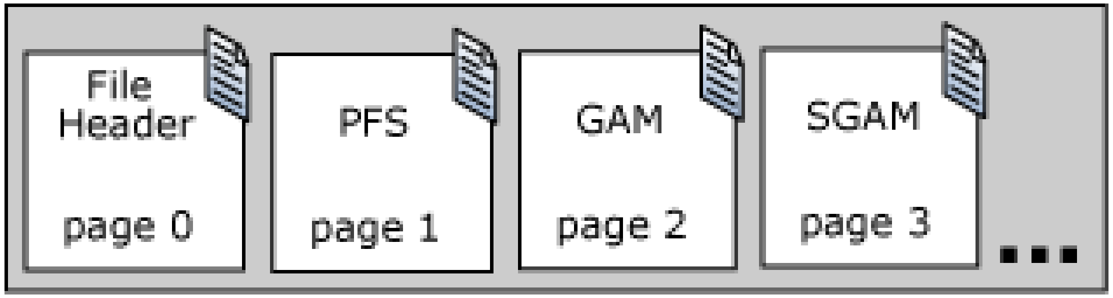
- Page Free Space (PFS) pages record the allocation status of each page, whether an individual page has been allocated, and the amount of free space on each page.
- The PFS has one byte for each page, recording whether the page is allocated, and if so, whether it is empty, 1 to 50 percent full, 51 to 80 percent full, 81 to 95 percent full, or 96 to 100 percent full.
- After an extent has been allocated to an object, the Database Engine uses the PFS pages to record which pages in the extent are allocated or free. This information is used when the Database Engine has to allocate a new page.

Tracking Free Space

- The amount of free space in a page is only maintained for heap and Text/Image pages. It is used when the 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

File structure

- A PFS page is the first page after the file header page in a data file (page number 1). This is followed by a GAM page (page number 2), and then an SGAM page (page 3). There is a PFS page approximately 8,000 pages in size after the first PFS page. There is another GAM page 64,000 extents after the first GAM page on page 2, and another SGAM page 64,000 extents after the first SGAM page on page 3.



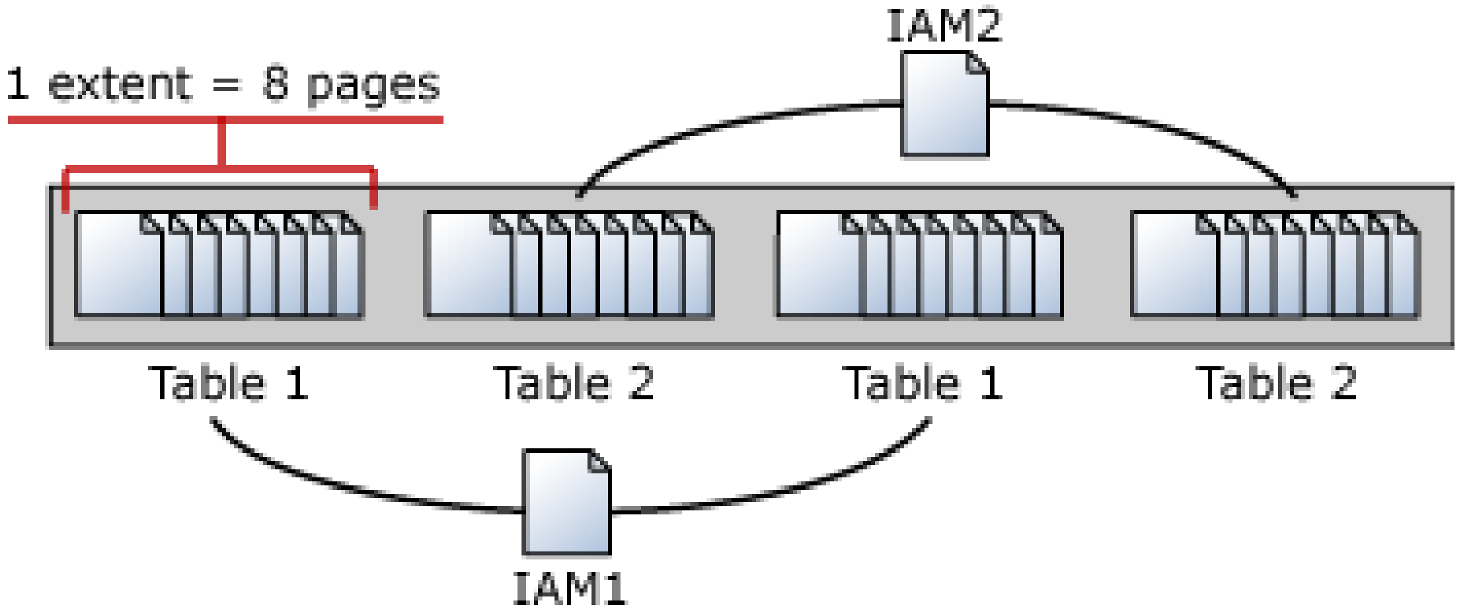
Space Used by Allocation Units

- An Index Allocation Map (IAM) page maps the extents in a 4-gigabye (GB) part of a database file used by an allocation unit
- An IAM page has 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.

IAM Pages

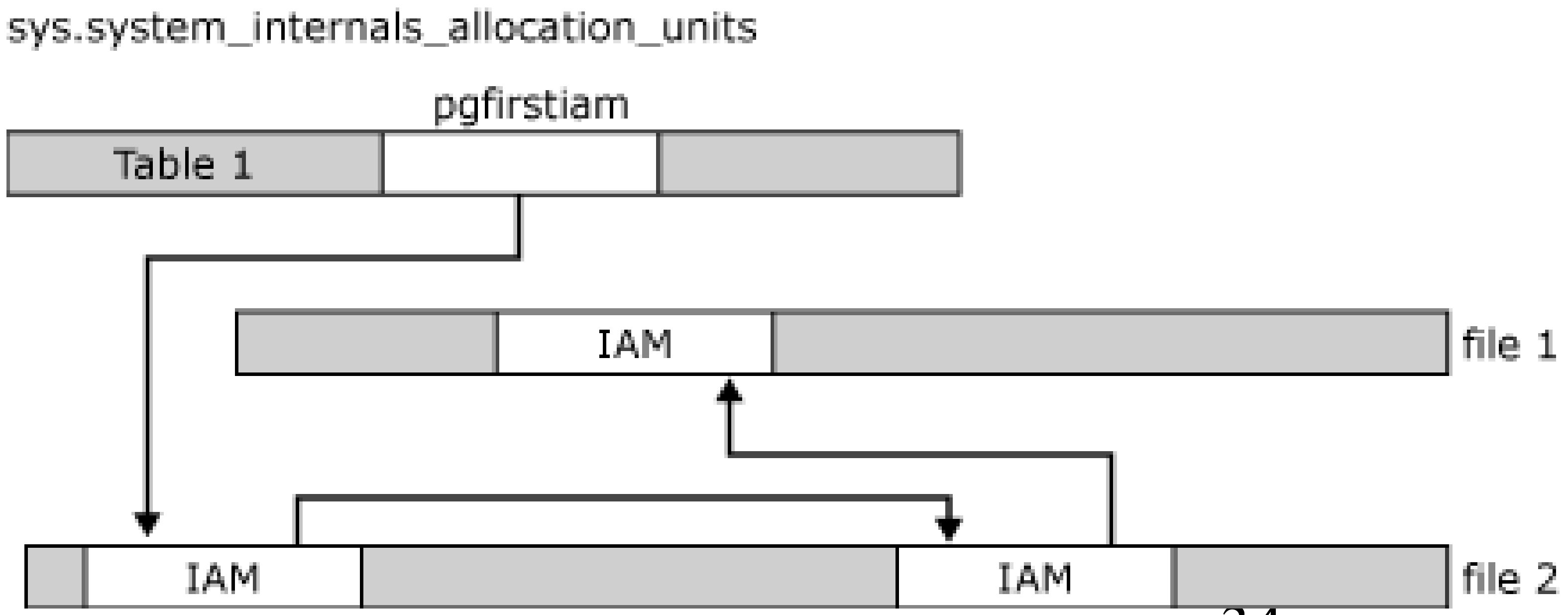
- An IAM page has a header that indicates the starting extent of the range of extents mapped by the IAM page.
- The IAM page also has a large bitmap in which each bit represents one extent. The first bit in the map represents the first extent in the range, the second bit represents the second extent, 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.

IAM Pages



IAM Pages

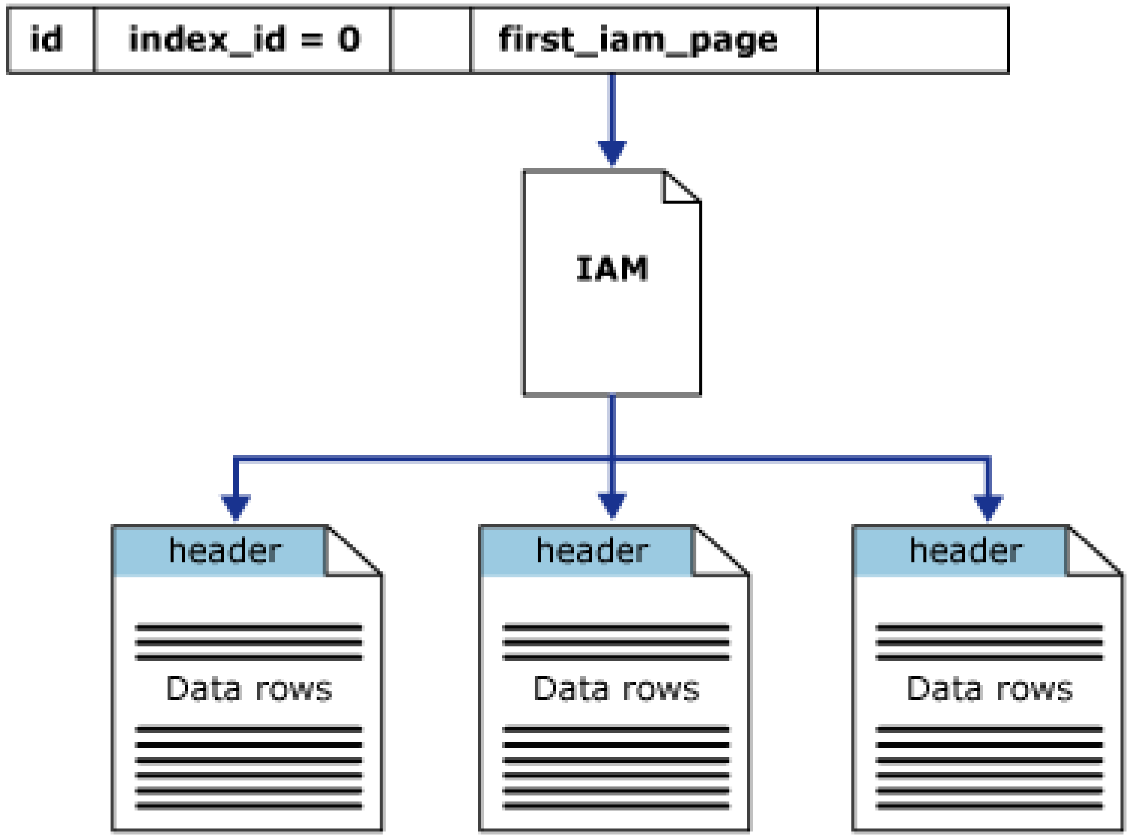
- IAM pages are allocated as required for each allocation unit and are located randomly in the file.
- The catalog view, **sys.system_internals_allocation_units** points to the first IAM page for an allocation unit.



Heap Organization

- The IAM pages are used to move through the heap. The only logical connection between data pages is the information recorded in the IAM pages.
- The column **first_iam_page** in the **sys.system_internals_allocation_units** catalog 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.
- 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.

Heap Organization



Space Allocation Algorithms

- When the SQL Server Database Engine has to insert a new row in a heap or Text/Image page and no space is available in the current page, it uses the IAM and PFS pages to find a page with sufficient space to hold the row.
- The Database Engine uses the IAM pages to find the extents allocated to the allocation unit. For each extent, the 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.

Space Allocation Algorithms

- For clustered tables, the insertion point of a new row is set by the index key. In this case, the search process previously described does not occur.
- The 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.

System Databases

- System databases store metadata regarding the databases available in the instance

System database	Description
master Database	Records all the system-level information for an instance of SQL Server.
msdb Database	Is used by SQL Server Agent for scheduling alerts and jobs.
model Database	Is used as the template for all databases created on the instance of SQL Server. Modifications made to the **model** database, such as database size, collation, recovery model, and other database options, are applied to any databases created afterward.
Resource Database	Is a read-only database that contains system objects that are included with SQL Server 2005. System objects are physically persisted in the **Resource** database, but they logically appear in the **sys** schema of every database.
tempdb Database	Is a workspace for holding temporary objects or intermediate result sets.

master Database

- Records all the system-level information for a SQL Server instance. This includes:
  - instance-wide metadata such as logon accounts, endpoints, linked servers, and system configuration settings, other databases and the location of those database files
  - initialization information for SQL Server. Therefore, SQL Server cannot start if the **master** database is unavailable. In SQL Server 2005, system objects are stored in the Resource database.

Physical files of the master DB

File	Logical name	Physical name	File growth
Primary data	master	master.mdf	Autogrow by 10 percent until the disk is full.
Log	mastlog	mastlog.ldf	Autogrow by 10 percent to a maximum of 2 terabytes.

Recomentations for the master DB

- Always have a current backup of the **master** database available.
- Back up the **master** database as soon as possible after the following operations:
  - Creating, modifying, or dropping any database
  - Changing server or database configuration values
  - Modifying or adding logon accounts
- Do not create user objects in **master**. Otherwise, **master** must be backed up more frequently.

Model Database

- When a CREATE DATABASE statement is issued, the first part of the database is created by copying in the contents of the **model** database. The rest of the new database is then filled with empty pages.
- If you modify the **model** database, all databases created afterward will inherit those changes. For example, you could set permissions or database options, or add objects such as tables, functions, or stored procedures.

Resource Database

- The **Resource** database is a read-only database that contains all the system objects that are included in SQL Server 2005.
- SQL Server system objects, such as **sys.objects**, are physically persisted in the **Resource** database, but they logically appear in the **sys** schema of every database. The **Resource** database does not contain user data or user metadata.
- The **Resource** database is not visible from Management Studio
- Can be used only in single user mode and for troubleshooting

tempdb Database

- It is a global resource that is available to all users connected to the instance of SQL Server
- It is used to hold the following:
  - Temporary user objects that are explicitly created, such as: global or local temporary tables, temporary stored procedures, table variables, or cursors.
  - Internal objects that are created by the SQL Server 2005 Database Engine, for example, work tables to store intermediate results for sorting.
- Operations within **tempdb** are minimally logged. This enables transactions to be rolled back. **tempdb** is re-created every time SQL Server is started so that the system always starts with a clean copy of the database.
- Temporary tables and stored procedures are dropped automatically on disconnect

Modifying System Tables

- SQL Server does not support users directly updating the information in system objects such as system tables, system stored procedures, and catalog views.
- To update the information one must use:
  - Administration utilities, such as SQL Server Management Studio.
  - SQL-SMO (Server Management Objects) API. This lets programmers include complete functionality for administering SQL Server in their applications.
  - Transact-SQL scripts and stored procedures. These can use system stored procedures and Transact-SQL DDL statements

Viewing System Database Data

- You should not code Transact-SQL statements that directly query the system tables,
- Instead, applications should obtain information from the system tables by using the following:
  - System catalog views
  - SQL-SMO
  - Windows Management Instrumentation (WMI) interface
  - Catalog functions, methods, attributes, or properties of the data API used in the application, such as ADO, OLE DB, or ODBC.
  - Transact-SQL system stored procedures and built-in functions.

System Catalog

- Is the collections of metadata, or data about the objects of databases, such as:
  - Tables, columns, indexes, views
- It is used by dynamic applications: applications that are not hard-coded to work with a specific set of tables and views must have a mechanism for determining the structure and attributes of the objects in any database to which they connect

Accessing the System Catalog

- SQL Server-based applications can access the information in the system catalogs by using the following:
  - Catalog views (recommended).
  - Information schema views (allow SQL-92 compatibility).
  - OLE DB schema rowsets.
  - ODBC catalog functions.
  - System stored procedures and functions.

Catalog Views

- All user-available catalog metadata is exposed through catalog views.
- They are the views of the master database
- They belong to the schema **sys**
- Examples:
  - **sys.partitions**
  - **sys.system_internals_allocation_units**
  - **sys.database_files**
  - **sys.tables**
  - **sys.views**

sys.partitions

- **sys.partitions** contains 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.

sys.partitions

- 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** in **sys.system_internals_allocation_units** column points to the chain of IAM pages that manage the pages in the LOB_DATA allocation unit.