Module 07 Notes

Database Management

# Module Overview

This module demonstrates three typical database management tasks; maintaining indexes, creating and testing databases backups, and configuring permissions.

# Required Software

* SQL Developer Edition (or equivalent)
* Visual Studio 2019 Community edition with the SSRS Extension
* SSMS

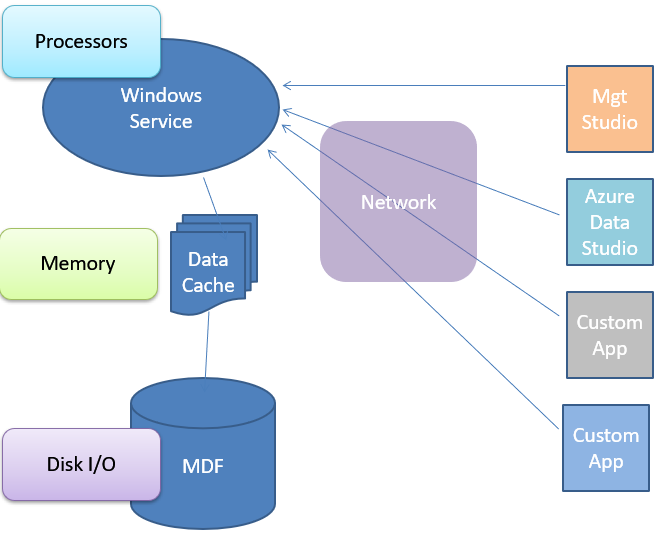
# Assignment

Each week you must perform an assignment. Let us review this week's assignment.

# SQL Server Architecture

To understand SQL Server management, you need to understand its basic architecture.

Like any software, SQL Server uses a combination of processors, memory, and disk input/output.



## Databases

SQL Server uses four standard databases and up to about 32 thousand user databases:

* Master - Configurations and Server Security
* Model - Starting Database Template
* MSDB - Configurations and monitoring of Microsoft's added features
* TempDB - Temporary objects

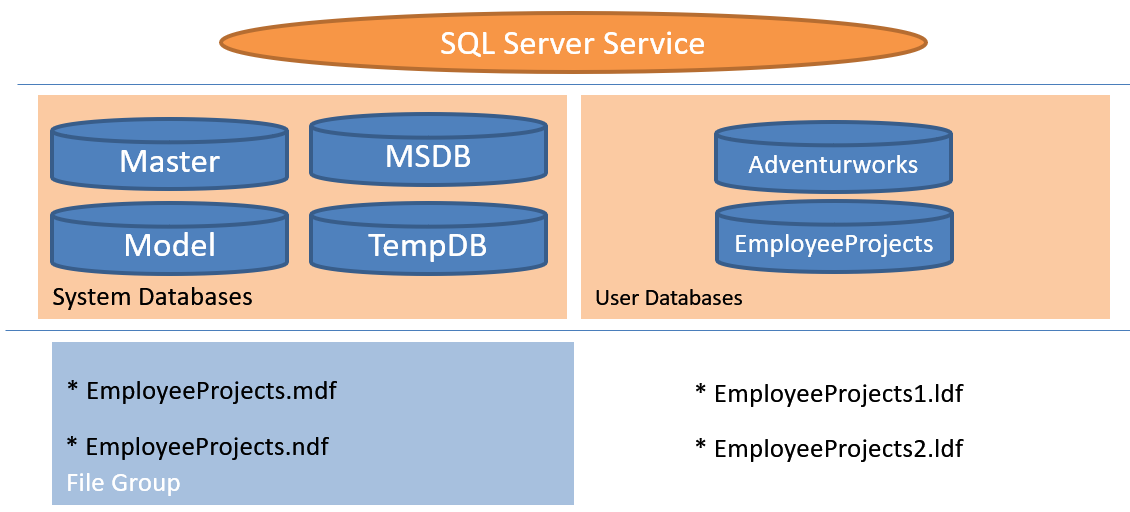
Each database has a Main data file (mdf) and a Log file (ldf), but more data and log files can be added.

### Database File Types

SQL Server databases have three types of files.

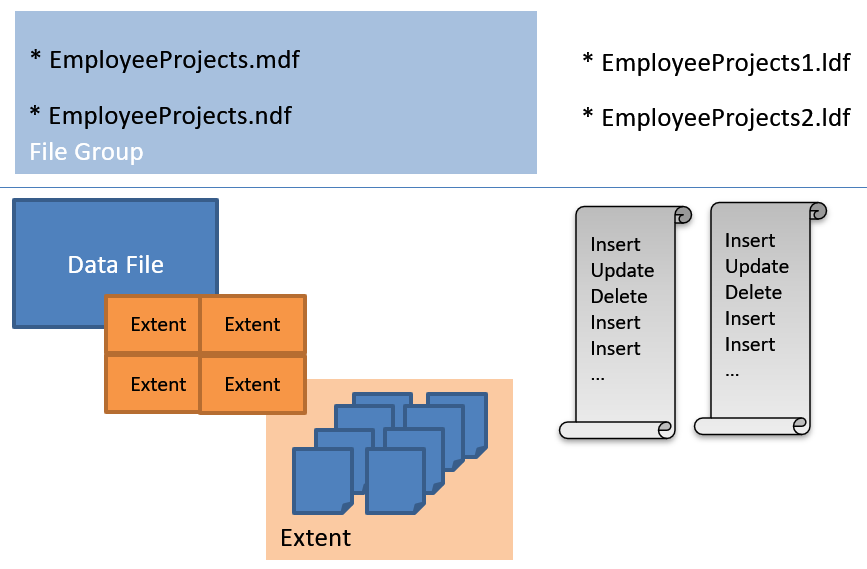
|  |  |
| --- | --- |
| ***Type*** | ***Desc*** |
| *Primary* | *Contains startup information for the database and points to the other files in the database. Every database has one primary data file. The recommended file name extension for primary data files is .mdf.* |
| *Secondary* | *Optional user-defined data files. Data can be spread across multiple disks by putting each file on a different disk drive. The recommended file name extension for secondary data files is .ndf.* |
| *Transaction Log* | *The log holds information used to recover the database. There must be at least one log file for each database. The recommended file name extension for transaction logs is .ldf.* |

[*https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-files-and-filegroups?view=sql-server-ver15*](https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-files-and-filegroups?view=sql-server-ver15)



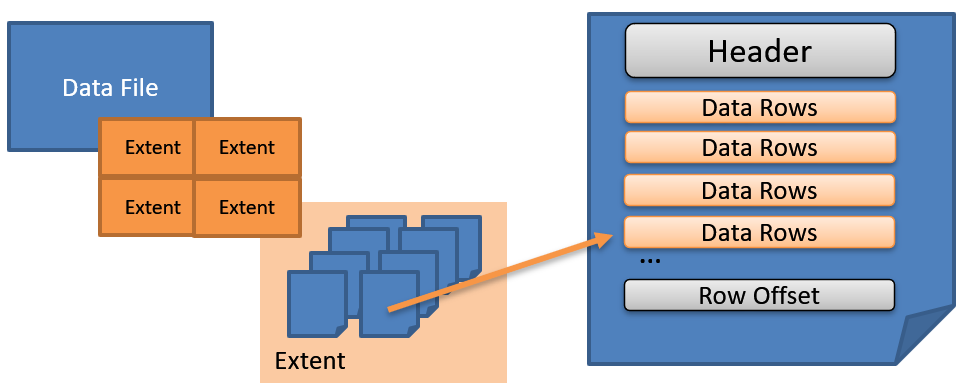
### Pages and Extents

Each data file contains Extents. Tables and index data is stored in **8 KB Pages** grouped into Extents **of 8 pages each**. However, each log file contains a linear list of changes to the database and not data pages.



**Note:** File Groups organize data files but **not log files**. **Each database has one Primary** filegroup, but you can create **more Filegroups to distribute data across hard drives**. This distribution is **similar to RAID 0 but at the database level instead of the OS level. *"Most databases will work well with a single data file and a single transaction log file."*** [***https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-files-and-filegroups?view=sql-server-ver15***](https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-files-and-filegroups?view=sql-server-ver15)***, 2020***

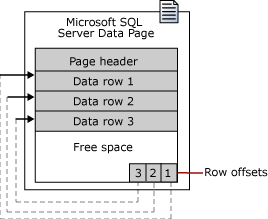
All **data pages contain a header, rows of data, and a footer**. The header describes the page, and the footer links pages together.



**"Data rows are put on the page serially, starting immediately after the header**

...

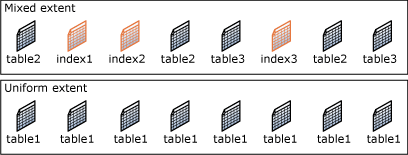
**Rows cannot span pages**, however portions of the row may be moved off the row's page so that the row can actually be very large. The maximum amount of data and overhead that is contained **in a single row on a page is 8,060 bytes** (8 KB). However, this does not include the data stored in the Text/Image page type. This restriction is relaxed for tables that contain varchar, nvarchar, varbinary, or sql\_variant columns.



...

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**.

...

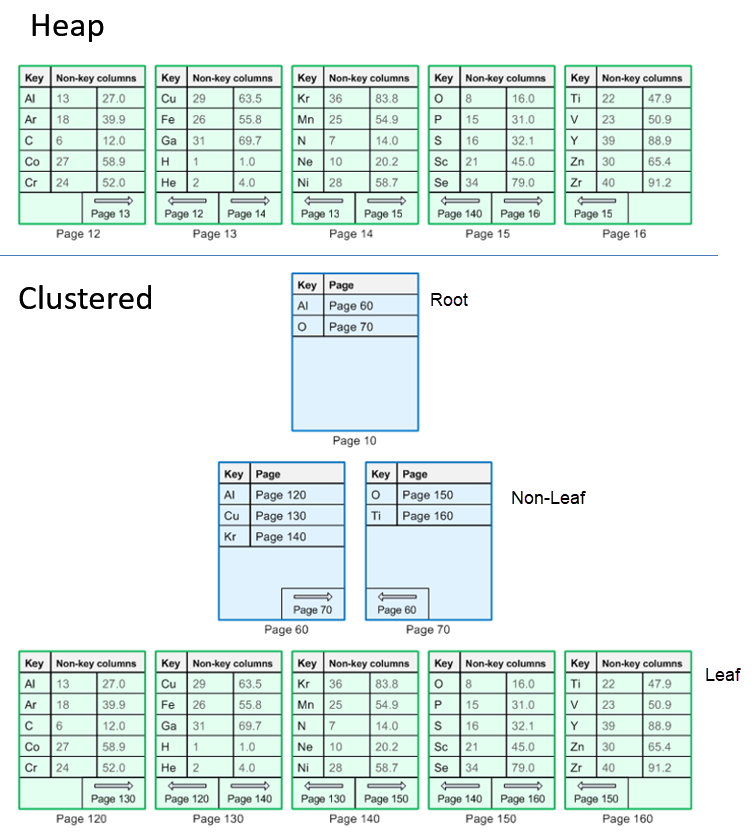


**A new table or index is generally allocated pages from mixed extents**. When the **table or index** grows to the point that it **has eight pages, it then switches to use uniform extents** for subsequent allocations. If you create an index on an existing table that has enough rows to generate eight pages in the index, all allocations to the index are in uniform extents.**"**

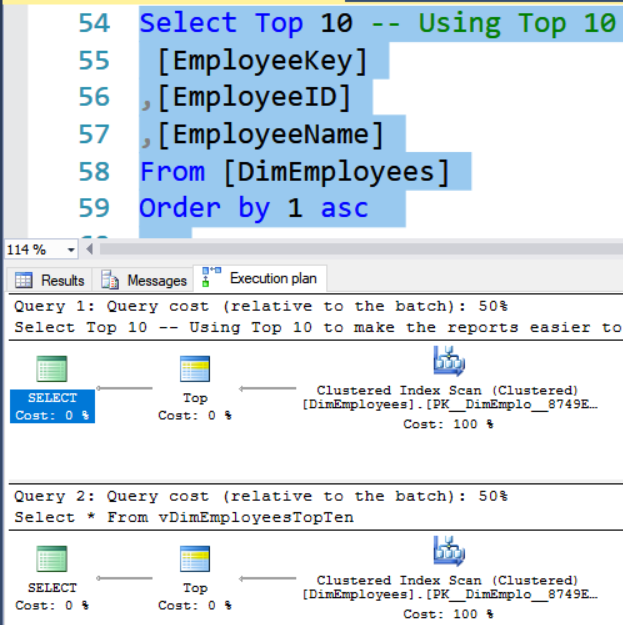
(<https://technet.microsoft.com/en-us/library/ms190969%28v=sql.105%29.aspx?f=255&MSPPError=-2147217396>, 2018)

### Heaps and Clusters

**Each table has one or more linked pages.** Tables default to a Heap, but become Clustered when a Clustered Index is placed on them. Indexes all have **a Root, Leaf, and Non-Leaf levels in a B-Tree Structure.**



When you review the Execution plan of a query, note how it depicts a Clustered Index scan.



# Demo 1: Heaps and Clusters

This demo looks at the differences between heaps and clusters.

# Indexing

Most tables will include **one or more indexes**. Indexes are **automatically** maintained whenever the table's data is modified.

## Types

There are two may types of indexes **Clustered and NonClustered.**

***"Clustered*** *indexes sort and store the data rows in the table or view based on their Key values. These are the columns included in the index definition. There can be* ***only one clustered index per table****, because the data rows themselves can be stored in only one order.*

*The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a* ***clustered index****, the table is called a* ***clustered table****. If a table has* ***no clustered index****, its data rows are stored in an unordered structure called* ***a heap****.*

***…***

***Nonclustered*** *indexes have a* ***structure separate from the data rows****. A nonclustered index contains the nonclustered index key values and each key value entry has* ***a pointer to the data row*** *that contains the key value.*

*The pointer from an index row in a nonclustered index to a data row is called a row locator. The structure of the row locator depends on whether the data pages are stored in a heap or a clustered table.* ***For a heap, a row locator is a pointer (RID)RR to the row. For a clustered table, the row locator is the clustered index key.****"* [*https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15*](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15)*, 2019*

## **Other Options**

There as several options that can be applied to an index.

### Unique

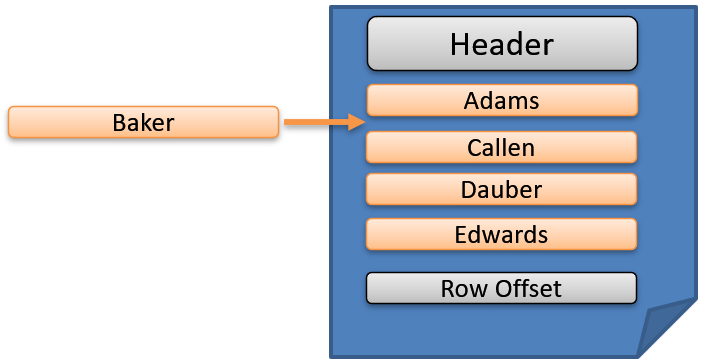
SQL Server **automatically** creates a unique index for a column (or columns) defined **with a primary key or unique constraint.**

### Composite

A composite **index uses more than one column in a table** for its index key. If your queries are on more than one table a Composite index can improve query performance. When all the columns returned by a query are included in an index, the **query is a "Covered Query" and the index is a "Covering Index."**

### Fill factor

Pages eventually become full, so new pages are added to the table or index. Adding a new page can sometimes **"Split" the page into two new pages**. A large number of Inserts, inserted into the middle of pages, cause **resource-intensive page splits**.

  
  
Fill factors are **applied when the index is created, rebuilt, or reorganized** to **allow** the data pages to be partially filled. This space **enables several row inserts before a page split occurs.**

*"Fillfactor" is a setting for indexes in SQL Server. When you create or rebuild an index, you can* ***tell SQL Server what percentage of each 8K data page*** *used in the*[*"leaf" level of the index*](https://www.brentozar.com/archive/2012/07/sql-server-index-terms/)***it should fill*** *up.*

*In other words, if you set a* ***90% fillfactor*** *when you rebuild a clustered index, SQL Server will try to* ***leave 10% of each leaf page*empty***. The empty space will be present on the page as it's written on disk, and it'll take up the same amount of space as it sits in memory.*

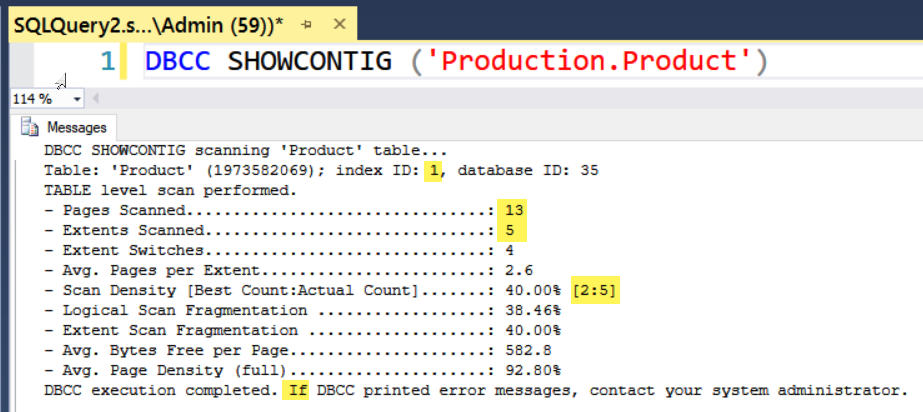
***By default, SQL Server uses a 100% fillfactor*** *and tries to fill up all the pages in indexes as close to full as it can. Depending on how many rows actually fit on the page, your mileage may vary."* [*https://www.brentozar.com/archive/2013/04/five-things-about-fillfactor/*](https://www.brentozar.com/archive/2013/04/five-things-about-fillfactor/)*, 2017*

# Demo 2: Indexes

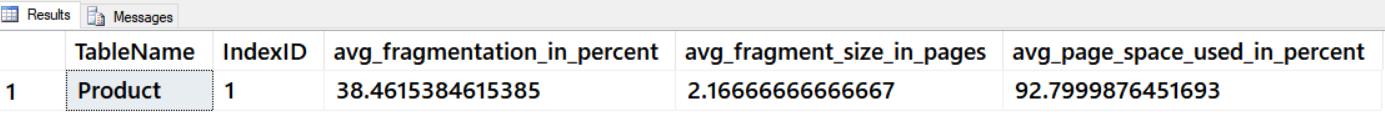
In this demo, we look at **how to create and get information about indexes**.

## Index Maintenance

**Indexes become fragmented over time**. A fragmented Index takes up more pages or extent then needed. The **DBCC SHOWCONTIG statement displays fragmentation information** on the data and indexes of a specified table.

DBCC SHOWCONTIG statement reports the **level of fragmentation in a table or specified index and how full the pages are.**

Microsoft has added new management views to SQL Server that give you similar information. The view **sys.dm\_db\_index\_physical\_stats dynamic is supposed to replace DBCC SHOWCONTIG** statement. It is **more accurate but also more difficult to read**. Here is a sample using the same table and index.



*"The value for* ***avg\_fragmentation\_in\_percent should be as close to zero as possible*** *for maximum performance.* ***However, values from 0 percent through 10 percent may be acceptable****. All methods of* ***reducing fragmentation****, such as* ***rebuilding, reorganizing, or re-creating****, can be used to reduce these values.*

***…***

*When an index is fragmented in a way that the fragmentation is affecting query performance, there are* ***three choices for reducing fragmentation****:*

***Drop and re-create the clustered index.***

*Re-creating a clustered index redistributes the data and results in full data pages. The level of fullness can be configured by using the FILLFACTOR option in CREATE INDEX. The drawbacks in this method are that* ***the index is******offline during the drop and re-create cycle****, and that the operation is atomic.* ***If the index creation is interrupted, the index is not re-created****. For more information, see CREATE INDEX (Transact-SQL).*

*Use* ***ALTER INDEX REORGANIZE****, the replacement for* ***DBCC INDEXDEFRAG****, to reorder the leaf level pages of the index in a logical order. Because this is an* ***online operation****, the index is available while the statement is running. The operation can also be interrupted without losing work already completed. The drawback in this method is that it* ***does not do as good a job of reorganizing*** *the data as an index rebuild operation, and* ***it does not update statistics****.*

*Use* ***ALTER INDEX REBUILD****, the replacement for* ***DBCC DBREINDEX****, to rebuild the index online or offline. For more information, see ALTER INDEX (Transact-SQL).*

*Fragmentation alone is not a sufficient reason to reorganize or rebuild an index. The* ***main effect of fragmentation is that it slows down page read****-ahead throughput during index scans. This causes slower response times. If the query workload on a fragmented table or index does not involve scans, because the workload is primarily singleton lookups, removing fragmentation may have no effect."* [*https://docs.microsoft.com/en-us/sql/relational-databases/indexes/reorganize-and-rebuild-indexes?view=sql-server-ver15*](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/reorganize-and-rebuild-indexes?view=sql-server-ver15)*, 2021*

# Demo 3: Index Maintenance

In this demo, we look at how to maintain an index using **Index Reorganize**.

# Database Backups

**A database is often the core of an organization's assets**. If the database is lost, you will have no way of which customer bought which products. You may need someone to retake inventory. You may not know what medicine a patient has been given. **Losing a database has destroyed many businesses**.

"To **minimize** the **risk** of catastrophic data loss, you need to back up your databases to **preserve** modifications to your **data** on a regular basis. A well-planned backup and restore strategy helps **protect** databases against **data loss caused by a variety of failures**. **Test** your strategy **by restoring** a set of backups and then recovering your database to prepare you to respond effectively to a disaster." ( <https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases>, 2017)

**Note: A database backup** will not only **capture** the data in the database, but **all the programming objects, users and roles, and permission settings**.

# Demo 4: Database Backups

This demo looks at how to Backup and Restore a database using some standard options.

## Backup Strategies

*"****Backing up and restoring data must be customized to a particular environment*** *and must work with the available resources. Therefore, a reliable use of backup and restore for recovery requires a backup and restore strategy. A well-designed backup and restore strategy* ***balances the business requirements for maximum data availability and minimum data loss, while considering the cost of maintaining and storing backups.***

*…*

*Designing an effective backup and restore strategy requires careful planning, implementation, and testing. Testing is required:* ***you do not have a backup strategy until you have successfully restored backups in all the combinations that are included in your restore strategy and have tested the restored database for physical consistency.****"* [*https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-ver15*](https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-ver15)*, 2018*

# Demo 5: Backup Strategies

This demo looks at how to Backup and Restore a database using some standard options.