**Primary vs. Secondary Files in Databases**

* **Primary Files**:
  + Store the main database records.
  + Organized by the **primary key**.
  + Only **one** primary file per database (e.g., .mdf in SQL Server).
  + Essential for database initialization.
* **Secondary Files:**
  + Optional additional storage when primary files are full.
  + Can have **zero or more** secondary files. (e.g., .ndf in SQL Server).
  + Used for performance optimization (spreading data across disks).
  + May store secondary indexes or additional data.

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

**Extension .mdf , .ndf and .ldf files**

| **File Extension** | **Stands For** | **Required?** | **Contains** |
| --- | --- | --- | --- |
| **.mdf** | **Master Database File** | **Yes (1 only)** | **Main data, system tables, metadata** |
| **.ndf** | **Next/Secondary Data File** | **No (0 or more)** | **Additional data, user objects** |
| **.ldf** | **Log Database File** | **Yes (1 or more)** | **Transaction logs for recovery/rollback** |

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

**Files and Filegroups**

**What are Files in a Database?**

A **file** is a physical structure on the disk used to store **database data or logs**.

There are 2 types of **data files** and 1 type of **log file**:

| **Type** | **File Extension** | **Purpose** |
| --- | --- | --- |
| Primary Data File | .mdf | Stores system tables, metadata, main data |
| Secondary File | .ndf | Stores additional user data (optional) |
| Log File | .ldf | Stores transaction logs |

So, **files** are just the actual containers on disk where data and logs are stored.

**What are Filegroups?**

A **filegroup** is a **logical grouping of one or more data files**.

**Types of Filegroups**

1. **Primary Filegroup**
   * Contains the **primary data file (**.mdf**)**.
   * All system tables are stored here.
   * If no other filegroup is specified, objects are stored here by default.
2. **User-Defined Filegroups**
   * Created by DBAs to **group**.ndf**files** logically.
   * Helps in:
     + **Partitioning large tables** across multiple disks.
     + **Improving I/O performance** by distributing workload.
     + **Backup/restore strategies** (can back up individual filegroups).

|  |  |
| --- | --- |
| **Component** | **Purpose** |
| .mdf | Primary data file (mandatory). |
| .ndf | Secondary data files (optional, improves performance). |
| .ldf | Transaction log (critical for recovery). |
| **Filegroups** | Logical groups of files for better management & performance. |

**Normalization vs. Mapping in Database Design**

**Normalization** is the process of organizing data in a database to **minimize redundancy** and **improve data integrity.**

**Goals of Normalization**

✔ **Eliminate duplicate data** (redundancy)  
✔ **Reduce anomalies** (insert, update, delete anomalies)  
✔ **Improve data integrity** (consistent and accurate data)  
✔ **Optimize storage** (efficient use of space)

|  |  |
| --- | --- |
| **Normal Form** | **Rule** |
| **1NF** | All attributes must be atomic (no repeating groups). |
| **2NF** | Must be in 1NF + no partial dependency (non-key columns depend on the whole primary key). |
| **3NF** | Must be in 2NF + no transitive dependency (non-key columns depend only on the primary key). |
| **BCNF** | Stricter than 3NF—every determinant must be a candidate key. |
| **4NF** | No multi-valued dependencies. |

**Output of Normalization**

* A **well-structured database** with **multiple related tables**.
* **Foreign keys** establish relationships.
* **Minimal redundancy** and **better query performance** (in OLTP systems).

**Output of Mapping**

* A **preliminary relational schema** (may still have redundancy).
* **Not necessarily normalized** (may need further refinement).
* **Direct translation** of ER diagrams into tables.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Mapping** | **Normalization** |
| **Purpose** | Converts ER model to tables. | Optimizes tables to reduce redundancy. |
| **Output** | Initial schema (may have redundancy). | Refined schema (minimal redundancy). |
| **When Applied?** | After ER modeling, before normalization. | After mapping, before implementation. |
| **Focus** | Structural translation (ER → Tables). | Data integrity & efficiency. |
| **Redundancy** | May still exist. | Eliminated (if fully normalized). |

**Denormalization**

**Denormalization** is the intentional process of **redundantly storing data** in a database to improve **read performance.**

**Key Characteristics**

✔ **Adds controlled redundancy** (duplicates data)  
✔ **Reduces JOIN operations** (speeds up queries)  
✔ **Common in data warehouses & reporting systems**  
✔ **Increases storage usage** (but storage is cheap today)

**When is Denormalization Needed in Business?**

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Why Denormalize?** | **Example** |
| **Analytics & Reporting** | Faster aggregations (fewer JOINs) | Storing sales data with customer names directly instead of linking tables. |
| **High-Read Systems** | Reduce database load for frequent reads. | E-commerce product listings with embedded category names. |
| **Real-Time Dashboards** | Speed up complex queries for live data. | Storing pre-computed metrics (e.g., monthly revenue per region). |
| **Caching & Materialized Views** | Avoid recalculating expensive queries. | Storing order history with customer details in one table. |
| **NoSQL & Big Data** | Optimized for horizontal scaling. | Document databases (MongoDB) storing nested data. |

**Pros & Cons of Denormalization**

| **Pros** | **Cons** |
| --- | --- |
| **Faster reads** (fewer JOINs) | **Redundant data** (increased storage) |
| **Simpler queries** (better for reporting) | **Update anomalies** (must sync duplicates) |
| **Better scalability** for read-heavy apps | **Complex maintenance** (triggers/ETL needed) |

**Clusterd in harddisk by primary key**

When a database table is clustered by primary key, it means the physical order of data rows on the hard disk is sorted based on the primary key values. This structure is implemented using a clustered index.

**Key Concepts**

1. **Clustered Index**
   * Determines the **physical storage order** of data.
   * Only **one clustered index per table** (usually on the primary key).
   * Example: If the primary key is EmployeeID, rows are stored in EmployeeID order on disk.
2. **How Data is Stored on Disk**
   * Instead of random placement, rows are **physically ordered** by the clustered key.
   * Similar to how a **phone book** is ordered by last name (clustered index on LastName).

**How Clustering Works on a Hard Disk**

**1. Without Clustering (Heap Table)**

* Rows are stored in **insertion order** (random physical placement).
* Finding a record requires a full scan or a separate index lookup.
* **Slower for range queries** (e.g., WHERE EmployeeID BETWEEN 100 AND 200).

**2. With Clustering (Clustered Index)**

* Rows are **physically sorted** by the primary key.
* **Faster for:**
  + Range queries (sequential disk reads).
  + ORDER BY operations on the clustered key.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Clustered Index** | **Non-Clustered Index** |
| **Storage Order** | Sorts & stores data rows physically. | Separate structure (like a book index). |
| **Number Allowed** | Only 1 per table. | Multiple per table. |
| **Speed** | Faster for range queries. | Faster for single-record lookups. |