# Why DBMS and What is it?

Imagine you run a railway ticket booking system using a file-based system. Every time a user searches for a ticket, the entire file (say 25GB) is loaded instead of fetching only the required 1KB of data. This leads to slow performance, high memory usage, and inefficient data retrieval. Also, multiple users booking tickets at the same time can cause data inconsistency. Also, the access management would be difficult for data inside the file or there can be small file issues.

* **Problem: File System Limitations**
  + **Slow Data Retrieval:** No indexing requires full file scan (O(n) complexity). E.g., fetching 1KB from a 20GB file needs reading the entire file.
  + **Manual Data Management:** Users must track file locations manually, leading to human errors.
  + **No Concurrency Control:** Two users editing the same file can cause overwrites and corruption.
  + **Data Redundancy:** No normalization, leading to duplicate records and wasted storage.
  + **Lack of Security & Access Control:** No granular permissions; entire files are accessible if shared.
* **Solution: DBMS Advantages**
  + **Optimized Retrieval:** Uses indexing and B-Trees for O(log n) lookup time instead of O(n).
  + **Abstracted Storage:** Users query data via SQL without knowing file locations.
  + **Concurrency Management:** Implements ACID properties to prevent data corruption.
  + **Data Normalization:** Eliminates redundancy using keys and constraints.
  + **Role-Based Access Control:** Provides fine-grained access, ensuring security.

# Which DBMS types to choose?

**1. Motivation: The Bookstore’s Growing Challenges**

* **Initial Approach:** File system (Excel, Text files)
* **Problems:**
  + Data redundancy & inconsistency
  + Slow data retrieval
  + Concurrency issues & lack of security

**2. Solution: RDBMS (Relational Database Management System)**

* **Examples:** MySQL, PostgreSQL, SQL Server
* **Why?**
  + Structured tables (Books, Customers, Sales)
  + Maintains consistency (ACID transactions)
  + Uses SQL for efficient queries

**3. New Challenges with Online Expansion**

* **High traffic → Read-heavy operations**
* **Flexible user data → Reviews, wishlists, preferences**
* **Distributed global operations**

**4. Other DBMS Solutions**

* **NoSQL (MongoDB, Cassandra, DynamoDB)**
  + Best for **semi-structured data** with schema flexibility
  + Handles large-scale **distributed storage**
* **Key-Value Store (Redis, DynamoDB, RocksDB)**
  + **Ultra-fast** key-based lookups
  + Used for caching & session management
* **Graph Database (Neo4j, ArangoDB)**
  + Best for analyzing **relationships** (e.g., recommendations, social networks)
* **Time-Series Database (InfluxDB, TimescaleDB)**
  + Optimized for **timestamped data** (e.g., IoT, stock prices)

**5. Conclusion: Choosing the Right DBMS**

* **RDBMS →** Structured data, consistency
* **NoSQL →** Scalable, flexible schema
* **Key-Value →** Speed, caching
* **Graph →** Relationship-heavy queries
* **Time-Series →** Fast ingestion of timestamped data

# General Architecture

**Two-Tier Architecture – Client & Server 🖥️➡️📊**

**Initially, database systems used a two-tier client-server model, where:**

* **The client (you, the data engineer) directly communicates with the database server to run queries.**
* **Example: Running SELECT \* FROM customers; on your SQL tool directly talks to the DBMS server to fetch data.**

**🚨 Problem?  
If multiple users access the database at once, performance degrades, and security is harder to manage.**

**Three-Tier Architecture – Adding a Middleman 🤝**

**To fix the issues of direct client-server communication, we introduced a middle layer:**

1. **Client (User Interface) – You write queries from an app or tool.**
2. **Application Server (Middle Layer) – Handles requests, security, caching before sending to DBMS.**
3. **Database Server – Actually stores and retrieves data.**

**🛠 Example:**

* **Instead of every user hitting the database directly, a web app or API acts as the middleman, improving scalability.**

**🚀 Why not four layers? Because three layers provide the right balance between flexibility, security, and performance without unnecessary complexity.**

**Three-Schema Architecture – A More Organized Library 📚**

**To manage databases better, we introduced three layers of abstraction:**

| **Schema Level** | **What it Represents** | **Analogy** |
| --- | --- | --- |
| **View Level (External)** | **What users see** | **Different bookshelves for different readers** |
| **Conceptual Level (Logical)** | **The database structure** | **The library’s master catalog** |
| **Physical Level** | **How data is stored** | **How books are arranged in storage rooms** |

**🛠 Example (as a Data Engineer)**

* **When you run SELECT \* FROM table; you’re in the View Level.**
* **When you create a table, you’re working in the Conceptual Level.**
* **When the database stores that table efficiently (indexes, partitions), it’s in the Physical Level.**

**Logical & Physical Data Independence – Why It Matters 🎯**

1. **Logical Data Independence → If we change the table structure (add a column), old queries should still work.**
2. **Physical Data Independence → If we change storage methods (heap → index), queries should still behave the same.**

**Why is this useful?  
Imagine if a library moved books to a better storage room, but readers could still find them without knowing the internal changes—that’s data independence!**

# 📚 Library Management System – Integrity Constraints in Action

**Imagine a library system where we store data about books, members, and borrowed books. Without proper constraints, the system can become inconsistent and unreliable. Let's walk through real-world issues and how integrity constraints fix them.**

* + 1. **Domain Constraint – Ensuring Valid Data**

**❌ Problem: What if someone enters a book with negative pages or a due date in the past?**

**✅ Solution: Domain constraints enforce valid data types and values.**

**Example:**

* **The Pages field must be a positive number.**
* **The Due\_Date must be a future date.**

**📝 SQL Code:**

CREATE TABLE Books (

Book\_ID INT PRIMARY KEY,

Title VARCHAR(255) NOT NULL,

Pages INT CHECK (Pages > 0), -- Prevents negative pages

Published\_Year INT CHECK (Published\_Year >= 1800) -- Books should have a valid year

);

**✅ Now, no book can have invalid pages or unrealistic publication years.**

* + 1. **Entity Integrity – Ensuring Uniqueness & Non-null Data**

**❌ Problem: What if a book is added without a Book\_ID, or two books have the same ID?**

**✅ Solution: Entity Integrity ensures each row is uniquely identifiable.**

**💡 Example:**

* **Each book must have a unique Book\_ID and cannot be NULL.**
* **Each member must have a unique Member\_ID.**

**📝 SQL Code:**

CREATE TABLE Members (

Member\_ID INT PRIMARY KEY, -- Ensures uniqueness

Name VARCHAR(100) NOT NULL -- Name cannot be NULL

);

**✅ Prevents duplicate or missing IDs, ensuring every record is unique.**

* + 1. **Referential Integrity – Maintaining Relationships**

**❌ Problem: What if a book is borrowed by a non-existent member?**

**✅ Solution: Referential Integrity ensures related records exist.**

**💡 Example:**

* **A borrowed book must belong to an existing member.**
* **If a member is deleted, their borrowed books must also be removed (CASCADE DELETE).**

**📝 SQL Code:**

CREATE TABLE BorrowedBooks (

Borrow\_ID INT PRIMARY KEY,

Book\_ID INT,

Member\_ID INT,

Borrowed\_Date DATE NOT NULL,

Due\_Date DATE NOT NULL,

FOREIGN KEY (Book\_ID) REFERENCES Books(Book\_ID),

FOREIGN KEY (Member\_ID) REFERENCES Members(Member\_ID) ON DELETE CASCADE

);

**✅ This prevents orphan records, ensuring every borrowed book is linked to a valid member.**

* + 1. **Key Constraints – Avoiding Duplicates**

**❌ Problem: What if two books have the same ISBN number?**

**✅ Solution: Key constraints ensure unique identifiers.**

**💡 Example:**

* **Each book should have a unique ISBN.**
* **No two members should have the same email.**

**📝 SQL Code:**

CREATE TABLE Books (

Book\_ID INT PRIMARY KEY,

Title VARCHAR(255) NOT NULL,

ISBN VARCHAR(13) UNIQUE -- ISBN should be unique

);

**✅ Now, no two books can have the same ISBN.**

* + 1. **Business Rule Constraints – Enforcing Library Rules**

**❌ Problem: What if a member borrows more than 5 books at a time?**

**✅ Solution: Business rule constraints enforce policies.**

**💡 Example:**

* **A member cannot borrow more than 5 books at a time.**

**📝 SQL Code:**

ALTER TABLE BorrowedBooks

ADD CONSTRAINT chk\_max\_books CHECK (

(SELECT COUNT(\*) FROM BorrowedBooks WHERE Member\_ID = BorrowedBooks.Member\_ID) <= 5

);

**✅ This ensures members cannot borrow more books than allowed.**

**📌 Summary: Why Are Integrity Constraints Important?**

**🚀 Data Accuracy → No negative pages or future-born books.  
🔑 Uniqueness → Every Book\_ID & Member\_ID is distinct.  
🔗 Relationships Intact → No missing members or books in borrowing records.  
📏 Business Rules Enforced → Library rules (like max borrow limit) are maintained.**

# Keys

| **Key Type** | **Definition** | **Example in Library System** | **Key Properties** |
| --- | --- | --- | --- |
| **Super Key** | A set of one or more columns that can uniquely identify a row. | {Book\_ID}, {ISBN}, {Book\_ID, Title} | 🔹 **Can have extra columns**  🔹 **Includes all possible unique keys** |
| **Candidate Key** | The minimal subset of a Super Key that uniquely identifies a row. | {Book\_ID}, {ISBN} | 🔹 **No unnecessary columns**  🔹 **Multiple Candidate Keys possible** |
| **Primary Key** | A chosen Candidate Key that uniquely identifies a row in the table. | {Book\_ID} | 🔹 **Must be unique**  🔹 **Cannot be NULL**  🔹 **Only one Primary Key per table** |
| **Alternate Key** | A Candidate Key that was **not selected** as the Primary Key. | {ISBN} (if {Book\_ID} is chosen as Primary Key) | 🔹 **Unique but not the Primary Key** |
| **Composite Key** | A key made of multiple columns to ensure uniqueness. | {Member\_ID, Book\_ID} in BorrowedBooks | 🔹 **Combination of columns**  🔹 **Used when a single column isn’t unique** |
| **Foreign Key** | A column that references the Primary Key in another table. | Book\_ID in BorrowedBooks referencing Book\_ID in Books | 🔹 **Establishes relationships**  🔹 **Ensures referential integrity** |
| **Unique Key** | Ensures column values are unique but allows NULL values. | Email in Members | 🔹 **Like a Primary Key but allows NULLs** |

**📚 Entity-Relationship (ER) Model: Structured Understanding**

**What is the ER Model?**

The **Entity-Relationship (ER) Model** is used to **design databases** before implementation. It represents:

* **Entities** → Objects (real-world things) that have attributes.
* **Attributes** → Properties describing an entity.
* **Relationships** → Connections between entities.

🔹 **Example (Library Management System)**

* **Entities**: Books, Members, BorrowedBooks
* **Attributes**: Book\_ID, Title, Author, Member\_ID, Name, Borrowed\_Date
* **Relationships**:
  + A Member **borrows** a Book
  + A Book **belongs to** a Category

# Elements of ER Model

**🔹 Entity (Table)**

An **Entity** is like a table in a database, representing **real-world objects**.

* **Strong Entity**: Has a **Primary Key** (e.g., Books, Members).
* **Weak Entity**: Depends on another entity and has a **Foreign Key** (e.g., BorrowedBooks).

**🔹 Attributes (Columns)**

Each entity has **attributes** that define its properties.

* **Simple Attribute**: Cannot be broken down (e.g., Name).
* **Composite Attribute**: Can be divided into sub-parts (e.g., Full Name → First Name, Last Name).
* **Derived Attribute**: Can be calculated (e.g., Age derived from DOB).
* **Multi-Valued Attribute**: Can have multiple values (e.g., Authors for a book).

**🔹 Relationship (Links Between Tables)**

A **relationship** represents how entities **interact** with each other.

* A **Member borrows Books** → BORROWS (Member\_ID → Book\_ID)

**🔹 Relationship Types & Cardinality**

Cardinality defines **how many instances** of one entity relate to another.

| **Relationship Type** | **Description** | **Example (Library)** |
| --- | --- | --- |
| **1:1 (One-to-One)** | Each entity has **exactly one** related entity. | A Library has **one** Librarian. |
| **1:M (One-to-Many)** | One entity relates to **many** others. | A Member can **borrow many** Books, but each Book is borrowed by **one** Member at a time. |
| **M:1 (Many-to-One)** | Many entities relate to **one** entity. | Many Books belong to **one** Category. |
| **M:M (Many-to-Many)** | Many entities relate to **many** others. | Books can be **borrowed by many** Members, and Members can borrow **multiple Books**. |

**📌 Many-to-Many requires a bridge table**  
Since SQL databases don’t support **direct M:M**, we use a **junction table** like BorrowedBooks(Member\_ID, Book\_ID).

**🔹 Conceptual vs. Logical vs. Physical Models**

* **ER Model (Conceptual Model)** → Abstract design (Entities, Attributes, Relationships).
* **Relational Model (Logical Model)** → Converts **ER diagrams to tables**.
* **Physical Model** → Defines actual **storage details** in the database.

# 📚 Normalization: A Step-by-Step Approach

**What is Normalization?**

Normalization is the process of **organizing a database** to:  
✔ **Reduce data redundancy** (no duplicate data).  
✔ **Avoid anomalies** (insertion, update, and deletion issues).  
✔ **Ensure data consistency** across related tables.

If **redundant data exists**, updating one record requires updating **all occurrences**, which leads to **update anomalies**. **Normalization solves this by breaking data into smaller, well-linked tables**.

**🌟 Normal Forms Explained with Library Example:** We will normalize a **Books table** step by step.

**🔥 Unnormalized Table (UNF) – Before Normalization**

📌 Consider a Books table where each row contains multiple authors:

| **Book\_ID** | **Title** | **Authors** | **Publisher** | **Category** |
| --- | --- | --- | --- | --- |
| 101 | DB Design Basics | John, Alice | Pearson | Database |
| 102 | Data Science Guide | Bob, Alice | O'Reilly | AI |
| 103 | Deep Learning | Eve, Charlie | Springer | AI |

💡 **Problems:**  
❌ **Multiple values in "Authors"** (not atomic).  
❌ **Redundancy** (Alice appears multiple times).  
❌ **Update issues** (changing Alice’s name requires updating multiple rows).

**🔹 First Normal Form (1NF)**

✔ **Rule:** Data must be **atomic** (no multiple values in a single cell).  
✔ **Fix:** Split multiple values into separate rows.

| **Book\_ID** | **Title** | **Author** | **Publisher** | **Category** |
| --- | --- | --- | --- | --- |
| 101 | DB Design Basics | John | Pearson | Database |
| 101 | DB Design Basics | Alice | Pearson | Database |
| 102 | Data Science Guide | Bob | O'Reilly | AI |
| 102 | Data Science Guide | Alice | O'Reilly | AI |
| 103 | Deep Learning | Eve | Springer | AI |
| 103 | Deep Learning | Charlie | Springer | AI |

💡 **Issues Fixed:**  
✔ **No multiple values in a single column**.  
❌ **Still redundant** (Book details repeat for each author).

**🔹 Second Normal Form (2NF)**

✔ **Rule:** No **partial dependency** (a **subset** of a **composite primary key** should not determine a non-prime attribute).  
✔ **Fix:** Split into two tables—**Books and Book\_Authors**.

**Books Table**

| **Book\_ID** | **Title** | **Publisher** | **Category** |
| --- | --- | --- | --- |
| 101 | DB Design Basics | Pearson | Database |
| 102 | Data Science Guide | O'Reilly | AI |
| 103 | Deep Learning | Springer | AI |

**Book\_Authors Table**

| **Book\_ID** | **Author** |
| --- | --- |
| 101 | John |
| 101 | Alice |
| 102 | Bob |
| 102 | Alice |
| 103 | Eve |
| 103 | Charlie |

💡 **Why is this better?**  
✔ **Authors are now stored separately, reducing duplication**.  
❌ **Still, some redundancy might exist** (like publishers repeating).

**🔹 Third Normal Form (3NF)**

✔ **Rule:** No **transitive dependency** (a **non-key attribute** should not determine another **non-key attribute**).  
✔ **Fix:** Remove Publisher’s dependency from Books and create a separate **Publisher table**.

**Books Table**

| **Book\_ID** | **Title** | **Category** | **Publisher\_ID** |
| --- | --- | --- | --- |
| 101 | DB Design Basics | Database | P01 |
| 102 | Data Science Guide | AI | P02 |
| 103 | Deep Learning | AI | P03 |

**Book\_Authors Table (Same as 2NF)**

**Publishers Table**

| **Publisher\_ID** | **Publisher\_Name** |
| --- | --- |
| P01 | Pearson |
| P02 | O'Reilly |
| P03 | Springer |

💡 **Why is this better?**  
✔ **No redundant publisher names**.  
✔ **Easier updates** (change publisher name in one place).