An Interactive Intro to Databases (DBMS)

Let's build a library, one concept at a time. 📚

What's a Database?

A **Database Management System (DBMS)** is a software tool that helps you organize, store, and retrieve information efficiently. Think of it as a super-powered digital filing cabinet.

Our real-life example will be a small **Library**. We need to keep track of our books and the people who borrow them.

1. The Building Blocks: Tables, Rows & Columns

In a database, information is stored in **tables**. A table is just a structured grid, like a spreadsheet.

- A **Column** defines a specific category of information (e.g., the book's title).
- A **Row** represents a single, complete item in the table (e.g., one specific book).

Here is our `Books` table:

Title (Column)	Author (Column)	Genre (Column)
The Hobbit	J.R.R. Tolkien	Fantasy
Dune	Frank Herbert	Sci-Fi
1984	George Orwell	Dystopian

The highlighted row above is one complete **record** or **row** for the book "Dune".

🧠 **Interactive Checkpoint:**
In the `Books` table, what information is in the 'Genre' column for the book '1984'?

▶ Show Answer

2. Describing Your Data: Data Types

Every column must have a **data type**. This tells the database what kind of information it will hold. It prevents you from, say, storing a book title where a publication year should be.

Common data types:

Data Type	Description	Example from our Library
`VARCHAR(n)` or `TEXT`	Text strings of varying length. `n` is the max characters.	'The Hobbit'
`INT` or `INTEGER`	Whole numbers.	1954 (Publication Year)
`DATE`	A date value.	'2025-09-21' (Date Borrowed)

3. Connecting the Dots: Keys & Relationships

Primary Keys 🔑

How do you uniquely identify a book if two books have the same title? We use a **Primary Key**. It's a column with a value that is unique for every single row.

Let's add an `ID` column to our tables. This is often an auto-incrementing integer.

Our updated `Books` table:

BookID (Primary Key)	Title	Author
1	The Hobbit	J.R.R. Tolkien
2	Dune	Frank Herbert

Relationships & Foreign Keys

Now, let's create a `Borrowers` table. It also needs a Primary Key.

BorrowerID (Primary Key)	FirstName	LastName
101	Alice	Smith
102	Bob	Jones

This leads to a **One-to-Many Relationship**: One Borrower can check out Many Books.
To track who borrowed which book, we need a third table, let's call it `Borrowals`.
This table uses **Foreign Keys** to link back to the other tables' Primary Keys.

Our `Borrowals` table:

BorrowalID (Primary Key)	BookID (Foreign Key)	BorrowerID (Foreign Key)	DateBorrowed
1001	2	101	2025-09-15
1002	1	102	2025-09-18

🧠 **Interactive Checkpoint:**
Based on the `Borrowals` table, who borrowed "The Hobbit"?

▶ Show Answer

4. Building the Structure: `CREATE TABLE`

Now let's see the actual code to create these tables. We use a language called **SQL (Structured Query Language)**.

Creating the `Books` table:

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(255),
    Author VARCHAR(100),
    Genre VARCHAR(50)
);
```

This SQL command defines the table name, its columns, the data type for each column, and specifies `BookID` as the Primary Key.

5. Asking Questions: Simple Queries (`SELECT`)

How do we get data out of our tables? We use the `SELECT` statement.

To get everything from the `Books` table:

```
SELECT * FROM Books;
```

(The `*` is a wildcard for "all columns".)

To get only the title and author of all books:

```
SELECT Title, Author FROM Books;
```

6. Filtering Your Results: The `WHERE` Clause

What if you don't want all the books, just specific ones? The `WHERE` clause lets you filter your results.

Find all books in the 'Fantasy' genre:

```
SELECT * FROM Books WHERE Genre = 'Fantasy';
```

Find the book with `BookID` 2:

```
SELECT Title FROM Books WHERE BookID = 2;
```

🧠 **Final Challenge:**
How would you write a query to find the `FirstName` and `LastName` of the borrower with `BorrowerID` 101?

▶ Show Answer

Congratulations! You've just learned the fundamentals of databases.