Database Design and Data Modeling Notes By Pradeep

These notes will guide you step-by-step from basic to advanced concepts, explained in very simple terms so anyone can understand. Perfect for studying and preparing for interviews.

I. Building Blocks of Data Modeling

Data modeling is like creating a **blueprint** for how information will be organized in a database. Just as you wouldn't build a house without a design, you wouldn't create a database without a model. It helps you plan how data will be stored and related to each other.

Key Concepts:

1. **Entity**:
- **What it is**: An **entity** is a "thing" you want to store information about.
- **Example**: In a school system, entities could be **Students**, **Teachers**, or **Courses**.
- **How to remember**: Entities are like nouns — people, places, or things.

2. **Attributes**:
- **What it is**: An **attribute** is a piece of information about an entity.
- **Example**: For a **Student**, attributes might include **Name**, **Age**, **Email**.
- **How to remember**: Attributes describe an entity, like adjectives describe nouns.

3. **Primary Key (PK)**:
- **What it is**: A **Primary Key** is a special attribute that uniquely identifies each record in a table. No two records will have the same primary key.
- **Example**: In a **Student** entity, the **StudentID** could be the primary key.
- **How to remember**: It's like a passport number or social security number — unique to each person.

4. **Foreign Key (FK)**:

- **What it is**: A **Foreign Key** links two entities together. It is an attribute in one entity that refers to the **Primary Key** of another entity.

- **Example**: In an **Enrollment** table, **CourseID** would be a foreign key that links to the **Course** entity.

- **How to remember**: It's like a reference. The foreign key in one table refers to the primary key in another.

5. **Relationships**:

- **What it is**: A relationship explains how two entities are connected.

- **Types of relationships**:

- **One-to-One (1:1)**: One student has one ID card, and each ID card belongs to only one student.

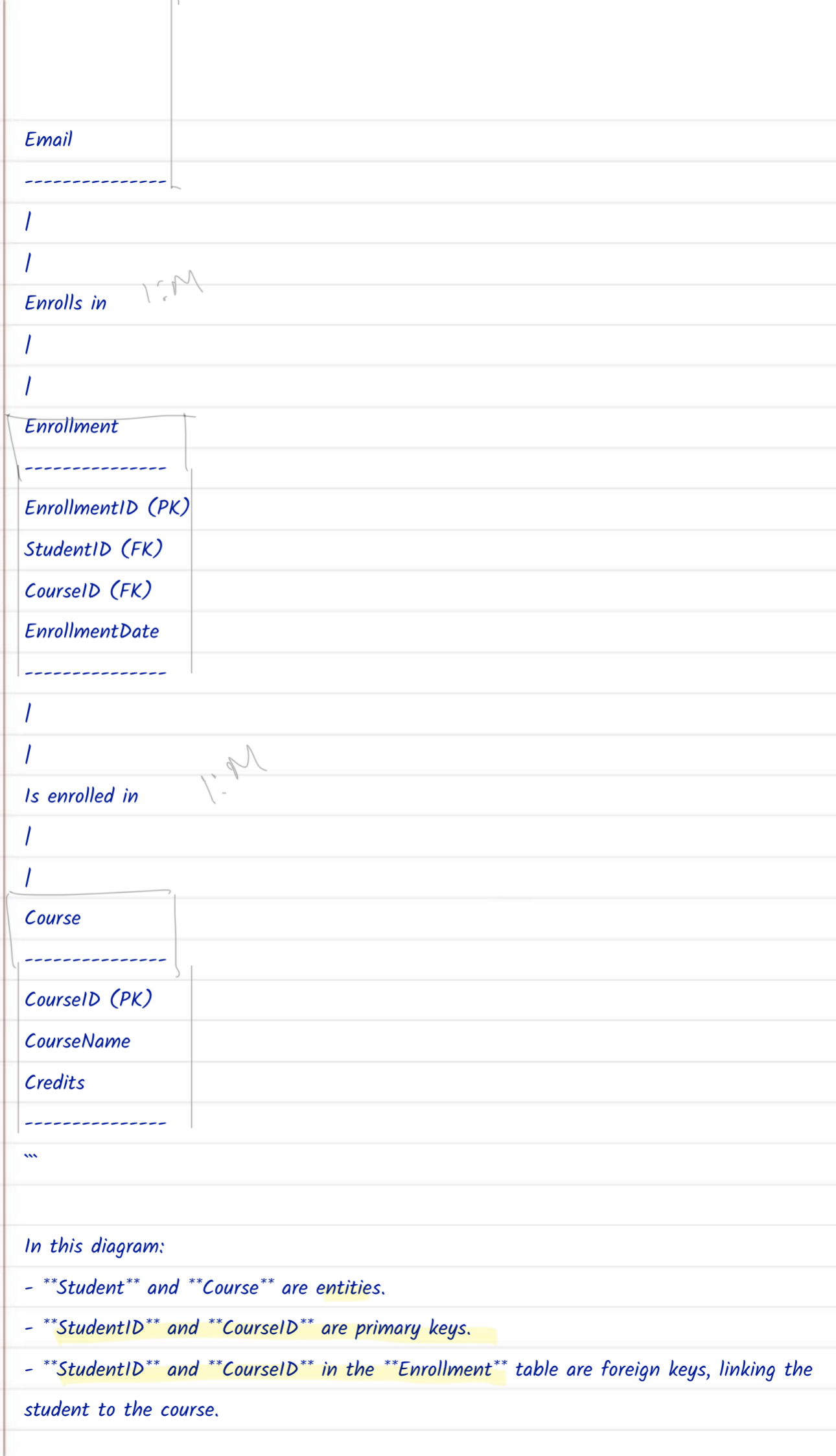
- **One-to-Many (1:M)**: One teacher can teach many courses, but each course is taught by only one teacher.

- **Many-to-Many (M:M)**: Many students can enroll in many courses, and each course can have many students. (This type of relationship requires a separate table, called a linking or join table.)

Example of a Simple Data Model (ERD):

Here is an **Entity-Relationship Diagram (ERD)** that shows the basic structure of a **school enrollment system**:

```
Student
---------------
StudentID (PK)
Name
Age
```



In this diagram:

- **Student** and **Course** are entities.
- **StudentID** and **CourseID** are primary keys.
- **StudentID** and **CourseID** in the **Enrollment** table are foreign keys, linking the student to the course.

---

2. Solving Problems with Data Modeling

Data modeling helps solve real-world problems by organizing data in a logical way. Here's how you can approach it:

Step-by-Step Approach to Problem Solving Using Data Modeling:

1. **Step 1: Identify Entities**:
- The first step is to figure out the main objects (entities) you want to keep track of.
- **Example**: In a **hospital** system, entities might be **Patients**, **Doctors**, and **Appointments**.

2. **Step 2: Define Attributes**:
- For each entity, figure out what details (attributes) you need to store.
- **Example**: For the **Patient** entity, attributes might include **PatientID**, **Name**, **Date of Birth**, **Address**, etc.

3. **Step 3: Establish Relationships**:
- Figure out how the entities relate to each other.
- **Example**: A **Doctor** can have many **Appointments**, and a **Patient** can have many **Appointments**. This is a **many-to-many** relationship, so we'd need a separate **Appointment** table.

4. **Step 4: Normalize Data**:
- **Normalization** is the process of organizing data to reduce duplication and ensure that it is consistent.
- **Example**: Instead of repeating patient information in every appointment, we store patient details in a **Patient** table and link it to the **Appointment** table using **PatientID**.

5. **Step 5: Create the Database Schema**:

- Once your model is ready, you can turn it into a **database schema**, which is the actual structure of the database. You will write this using SQL (Structured Query Language).

Normalization

Normalization is a process to **organize data** into multiple related tables to reduce redundancy and ensure accuracy. There are three main levels of normalization:

1. **First Normal Form (1NF)**:
- **What it means**: Data is stored in tables where each column contains atomic (indivisible) values, and each column contains values of a single type.
- **Example**: You wouldn't store a list of emails in one column. Instead, each email would be in its own row.

2. **Second Normal Form (2NF)**:
- **What it means**: The table is in **1NF**, and every attribute is fully dependent on the primary key.
- **Example**: In a **student grades** table, the grade should depend on both **StudentID** and **CourseID**, not just one.

3. **Third Normal Form (3NF)**:
- **What it means**: The table is in **2NF**, and all the non-primary key attributes depend only on the primary key, not on each other.
- **Example**: In a **Student** table, the **City** and **PostalCode** should be moved to a separate table if they depend on **Address** and not on **StudentID**.

---

3. Advanced Concepts in Data Modeling

Once you've mastered the basics, here are some more advanced concepts:

1. Normalization vs. Denormalization
- **Normalization**: Organize data into multiple smaller tables to avoid duplication and ensure accuracy. This is useful when you need to frequently update the data.
- **Denormalization**: Combine tables into fewer, larger tables to improve the speed of data retrieval (reading). This is used when fast reads are more important than data updates, such as in big data systems.

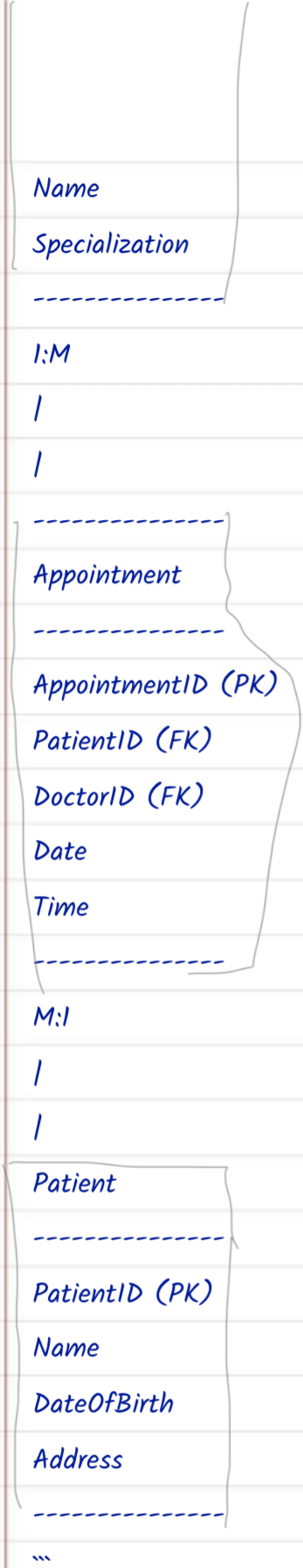
2. Cardinality
- **Cardinality** refers to the type of relationship between two entities:
- **One-to-One (1:1)**: A record in one table is related to only one record in another table.
- **One-to-Many (1:M)**: One record in a table is related to many records in another table.
- **Many-to-Many (M:M)**: Many records in one table relate to many records in another table. This is often handled by creating a **join table**.

3. Indexing
- **Indexing** is a way to speed up searches in a database by allowing the database to find rows faster, just like using an index in a book.
- **Primary Index**: Automatically created on the **Primary Key**.
- **Secondary Index**: Created on other columns for faster searching.

Advanced Example (Hospital System ERD):

Here is a more advanced example of a **Hospital System**:

```
Doctor
---------------
DoctorID (PK)
```




```

- **Doctor** and **Patient** are connected through the **Appointment** table.
- **AppointmentID** is the primary key in the **Appointment** table, and **DoctorID** and **PatientID** are foreign keys.

---
```

4. Optional Assignment: Build a Data Model for an Online Bookstore

Let's walk through a **data model for an online bookstore**.

Entities:
- **Customer**: Stores information about customers.
- **Order**: Stores details about customer orders.
- **Book**: Stores information about the books.
- **OrderItem**: Connects **Order** and **Book** in a **many-to-many** relationship.
- **Author**: Stores information about authors.

Attributes:
- **Customer**: CustomerID, Name, Email.
- **Order**: OrderID,

OrderDate, CustomerID (FK).
- **Book**: BookID, Title, Price, AuthorID (FK).
- **OrderItem**: OrderItemID, OrderID (FK), BookID (FK), Quantity.
- **Author**: AuthorID, Name.

Relationships:
- **Customer** places multiple **Orders** (1:M).
- **Order** contains multiple **Books** and **Books** appear in multiple **Orders** (M:M).
- **Author** writes many **Books** (1:M).

Example ERD (Online Bookstore):

```
Customer
---------------
CustomerID (PK)
Name
```

Email

1:M

|

Order

OrderID (PK)

OrderDate

CustomerID (FK)

M:M

Contains

|

OrderItem

OrderItemID (PK)

OrderID (FK)

BookID (FK)

Quantity

M:1

|

Book

BookID (PK)

Title

Price

AuthorID (FK)

|

1:M

|

Author

AuthorID (PK)

Name

```

---

Different stages of **data modeling** — **conceptual**, **logical**, and **physical**.

---

### **Data Modeling Workflow: Conceptual, Logical, and Physical Models**

Data modeling typically has **three stages**:

1. **Conceptual Data Model** (High-level overview)
2. **Logical Data Model** (More detail, but still technology-agnostic)
3. **Physical Data Model** (Detailed blueprint for the database)

Each stage provides more detail and is a refinement of the previous one. Let's look at each in detail.

---

### **1. Conceptual Data Model (What are the things?)**

- **What it is**: The **Conceptual Data Model** is the **simplest and broadest** model. It's like a rough sketch that shows the **main entities** and their **relationships** without going into much detail.
- **Goal**: To show **what** data needs to be stored, without worrying about how it will be
```

stored.

- **Who uses it**: Business stakeholders, project managers, and analysts who are more focused on **what the system should capture**.

Key Features:

- **Entities**: Defines the **main objects** in the system.
- **Relationships**: Shows **how entities are related** to each other.
- **Attributes**: In some cases, may include **basic attributes** (not too detailed).

Example:

For an **e-commerce system**, the **Conceptual Data Model** might identify these entities:

- **Customer**
- **Order**
- **Product**
- **Payment**
- **Cart**

And the relationships:

- **Customer** places **Orders**.
- **Orders** contain **Products**.
- **Customers** add **Products** to **Cart**.
- **Orders** have **Payments**.

---

2. Logical Data Model (How are things related and described?)

- **What it is**: The **Logical Data Model** adds more **details** to the conceptual model. It still focuses on **business logic** and **rules**, but it begins to define **specific attributes** and **relationships** in more depth.
- **Goal**: To define **how** the data is structured and what kind of **relationships** exist, still without focusing on the actual implementation (i.e., no specific database technology yet).

- **Who uses it**: Data architects and analysts who are designing the system's structure before it's built.

Key Features:
- **Entities**: Now more **detailed** with a focus on **attributes** and **unique identifiers**.
- **Attributes**: Includes **specific attributes** for each entity, including things like data types (e.g., text, number).
- **Relationships**: Clearly defined **cardinality** (e.g., one-to-many, many-to-many).

Example (E-commerce system):
For an **e-commerce system**, the **Logical Data Model** could look like this:

1. **Entities**:
- **Customer**: CustomerID, Name, Email, Address
- **Order**: OrderID, OrderDate, Status, CustomerID (FK)
- **Product**: ProductID, ProductName, Price, Stock
- **Payment**: PaymentID, PaymentDate, Amount, OrderID (FK)
- **Cart**: CartID, CustomerID (FK), ProductID (FK), Quantity

2. **Relationships**:
- **One-to-Many**: One **Customer** can place many **Orders**.
- **Many-to-Many**: Many **Orders** can contain many **Products**, so a linking entity like **Cart** is used to manage this.
- **One-to-One**: Each **Order** is associated with one **Payment**.

Diagram for Logical Model:
At this stage, you start defining entities and their attributes, as well as the relationships between them with more clarity, but still **no physical structure** or technology-specific details.

---

3. Physical Data Model (How will it be built?)

- **What it is**: The **Physical Data Model** is the **most detailed** model. It focuses on **how the data will actually be stored in a database**. This model includes **technology-specific details**, like what kind of database will be used (SQL, Oracle, etc.), and considers **performance, indexing**, and **storage**.
- **Goal**: To create the **actual blueprint** for the database. This is the step where you decide how the database tables, columns, keys, and relationships will be physically implemented.
- **Who uses it**: Database engineers, developers, and administrators who are responsible for building and maintaining the database.

Key Features:
- **Tables**: Entities are turned into actual **database tables**.
- **Columns**: Attributes are now **table columns** with specific data types (e.g., integer, varchar).
- **Primary and Foreign Keys**: Clear identification of **primary keys (PK)** and **foreign keys (FK)** for each table.
- **Indexes**: Plan for how to **optimize** the database (for example, adding indexes for faster search).

Example (E-commerce system):
For an **e-commerce system**, the **Physical Data Model** might look like this:

1. **Tables**:
- **Customer** Table:
- `CustomerID` (PK): INTEGER
- `Name`: VARCHAR(100)
- `Email`: VARCHAR(100)
- `Address`: VARCHAR(250)
- **Order** Table:
- `OrderID` (PK): INTEGER

- `OrderDate`: DATE
- `Status`: VARCHAR(20)
- `CustomerID` (FK): INTEGER (linked to **Customer** table)
- **Product** Table:
- `ProductID` (PK): INTEGER
- `ProductName`: VARCHAR(100)
- `Price`: DECIMAL(10,2)
- `Stock`: INTEGER
- **Payment** Table:
- `PaymentID` (PK): INTEGER
- `PaymentDate`: DATE
- `Amount`: DECIMAL(10,2)
- `OrderID` (FK): INTEGER (linked to **Order** table)
- **Cart** Table:
- `CartID` (PK): INTEGER
- `CustomerID` (FK): INTEGER (linked to **Customer** table)
- `ProductID` (FK): INTEGER (linked to **Product** table)
- `Quantity`: INTEGER

2. **Indexes**:
- Add an index on **CustomerID** for faster lookups on the **Customer** table.
- Add an index on **OrderID** in the **Payment** table to speed up payment lookups for orders.

Summary of the Three Data Models:

1. **Conceptual Data Model**:
- Focus: What data needs to be stored?
- Entities: Broad, without many details.
- Audience: Business stakeholders and analysts.

2. **Logical Data Model**:

- Focus: How are entities and attributes structured?

- Entities: More detailed, includes attributes and relationships.

- Audience: Data architects, analysts.

3. **Physical Data Model**:

- Focus: How will it be implemented in a database?

- Entities: Detailed database tables, columns, keys, and indexing strategies.

- Audience: Database engineers and developers.

Visualizing the Difference

- **Conceptual**: Just shows that a **Customer** places **Orders**, and an **Order** contains **Products**.

- **Logical**: Defines that the **Customer** has a **CustomerID**, **Name**, and **Email** and is related to the **Order**, which has attributes like **OrderDate** and **OrderID**.

- **Physical**: Specifies that the **CustomerID** is an **INTEGER** column, **Email** is a **VARCHAR(100)** column, and the **OrderID** is a **foreign key** in the **Payment** table.

Interview Preparation Tips

- **Understand each concept** thoroughly so you can explain it simply.

- **Practice drawing ERDs** by hand or on a whiteboard.

- Be able to **differentiate between normalization and denormalization**.

- Be comfortable using real-world examples like **school management systems**, **e-commerce websites**, and **hospital databases**.

Few quick links:

UBER DB: https://www.geeksforgeeks.org/how-to-design-a-database-for-uber/?ref=ml_lbp

E-COMMERCE: https://www.geeksforgeeks.org/how-to-design-a-relational-database-for-e-commerce-website/

HOSPITAL MANAGEMENT : https://www.geeksforgeeks.org/how-to-design-er-diagram-for-a-hospital-management-system/