Module 5:

Database Essentials & Relational Model

What is Database?

A database is a structured collection of related data that represent some real world entities and are organized for efficient retrieval, storage, and management.

What is Data?

**Data** is a **raw fact or figure** that has no meaning by itself but can be processed to produce **useful information**.

What is information?

Information is processed and organized data that provides meaningful context, insight or knowledge.

What is a DBMS (Database management System)?

A **DBMS** is **software** that is used to **store**, **manage**, **retrieve**, and **organize** data in databases efficiently.

What is RDBMS (Relational Database management System)?

**RDBMS** is a **type of DBMS** that **stores data in tables** (also called **relations**) and uses **SQL (Structured Query Language)** to manage it.

Problem in storing data using a file system.

1. Unstructured data, multiple formats.(.txt, .mp4, etc)
2. Data redundancy
3. Data inconsistency
4. No concurrency protocol
5. Security issue
6. Access complication

Types of Database.

1. Relational -> mySql, postGrace, SQLite, SQL Server
2. Document -> MongoDB, amazon DynamoDb
3. Key value -> redis

Table /Relation

column/attribute

constraint/domain

Degree(collection of column)

Rows/Tuples/Records

Cardinality(collection of rows)

What is key?

Key: A key in a relational database is a field or combination of fields that uniquely identifies a record in a table. Some important keys..

* + Super key
  + Candidate key
  + Alternate key
  + Composite key
  + Foreign key
  + Primary key

**Super key**

What is a super key?

a set of one or more attributes (columns) that can uniquely identify each tuple (row) in a relation (table)

Super key:

* Attribute or set of attribute by which we can identify each row uniquely
* Could be a single attribute or a set of attributes
* Could have null values in the set
* It actually a superset

| StudentID | Name | Email |
| --- | --- | --- |
| 101 | Alice | alice@gmail.com |
| 102 | Bob | bob@gmail.com |

#### **Super Keys for this table could be:**

* {StudentID}
* {Email}
* {StudentID, Name}
* {StudentID, Email}  
   👉 All of these **can uniquely identify** a student, so they are super keys.

**Candidate key**

What is Candidate key:

A **Candidate Key** is a **minimal set of attributes** that can **uniquely identify a row** in a table.

### **What does "Minimal Set" mean?**

A **minimal set** of attributes means:

You have the **smallest number of columns** needed to **uniquely identify** a row — and if you remove **any one attribute**, it **won’t work anymore**.

| Roll | Name | Email |
| --- | --- | --- |
| 101 | Alice | alice@gmail.com |
| 102 | Bob | bob@gmail.com |

#### **{Roll}**

#### It uniquely identifies each row.

#### It’s minimal — only one attribute. So it's a **candidate key**.

#### **{Roll, Email}**

#### It also uniquely identifies each row → (it's a **super key**)

#### BUT you don’t need both — **Roll alone is enough**. So it's **not minimal**, **not a candidate key**.

#### 

It is a **super key with no unnecessary attributes**.

### **Properties of Candidate Key:**

1. **Uniqueness** – Must uniquely identify each row.
2. **Minimality** – No attribute can be removed from it and still keep uniqueness.
3. **No NULLs allowed** – Every row must have a value.

Candidate key:

* Super key whose proper subset is not a super key
* Also called Minimal Super key
* Potential Primary Key: Form the candidate keys. One is chosen as the primary key. However, all candidate keys are potential choices for the primary key.

| StudentID | Name | Email |
| --- | --- | --- |
| 101 | Alice | alice@gmial.com |
| 102 | Bob | bob@gmail.com |
| 103 | Carol | carol@gmail.com |

#### **Possible Candidate Keys:**

* {StudentID} → ✅ Unique for each student
* {Email} → ✅ Unique for each student

#### **Not a Candidate Key:**

* {StudentID, Email} → ❌ This is a **super key**, but **not minimal** (contains unnecessary attributes).

Extra:

* One of the **candidate keys** is chosen as the **Primary Key**.
* Others may become **Alternate Keys**.

**\*\*\*Proper subset**: A smaller set of attributes taken from the super key.

Let’s say the following:

* {ID, Email} → ✅ Super key (it uniquely identifies rows)
* But:  
  + {ID} → ✅ Still a super key
  + {Email} → ✅ Still a super key

So:

* {ID, Email} is **not** a **candidate key**, because its **subsets are also super keys**

Now consider:

* {ID} → ✅ Super key
* Try removing any part — you can’t; it's a **minimal** set  
   👉 Therefore, {ID} is a **candidate key**

**Primary key**

What is a primary key?

A primary key is an attribute or a set of attributes that uniquely identifies each row in a table.

**Primary key:**

* From the candidate key, one key is chosen as the primary key for the table. The primary key is a specific candidate key that is selected as the main identifier for the records in that table
* Should be unique, not null and stable

| StudentId | Name | Email |
| --- | --- | --- |
| 101 | Alice | alice@gmail.com |
| 102 | Bob | bob@gmail.com |

If we define StudentID as the **Primary Key**, it means:

* No two students can have the **same StudentID**.
* StudentID can **never be NULL**.

🔍 Key Properties of Primary Key:

| **Property** | **Description** |
| --- | --- |
| **Uniqueness** | No duplicate values allowed |
| **Not NULL** | Must always have a value |
| **Single per table** | A table can have only **one** primary key |
| **Chosen from candidate keys** | You select one best candidate key |

✅ Primary Key vs Super Key vs Candidate Key:

| **Key Type** | **Uniqueness** | **Minimal** | **NULL Allowed** | **Count per Table** |
| --- | --- | --- | --- | --- |
| Super Key | ✅ | ❌ | Can include NULL | Many |
| Candidate Key | ✅ | ✅ | ❌ | Many |
| Primary Key | ✅ | ✅ | ❌ | **Only One** |

**Alternate key**

What is an Alternate key?

An **Alternate Key** is any **candidate key** in a table **that is not chosen** as the **Primary Key**.

It's a "backup" unique key that could have been the primary key, but wasn’t selected.

Alternate key:

* Candidate keys which were not chosen as primary key

| StudentId | Phone | Email |
| --- | --- | --- |
| 101 | 1234567890 | alice@gmail.com |
| 102 | 0987654321 | bob@gmail.com |

#### **Candidate Keys:**

* {StudentID}
* {Email}
* {Phone}

Let’s say:

* You choose StudentID as the **Primary Key**.

Then:

* Email and Phone become **Alternate Keys** — because they also uniquely identify rows but were **not chosen** as the primary key.

**Composite key**

### **What is a Composite Key?**

A **Composite Key** is a **primary key** (or candidate key) that is made up of **two or more columns** (attributes) that together **uniquely identify a row** in a table.

🔸 A **single column** is not enough to ensure uniqueness, but **the combination is**.

| StudentId | CourseId | Grade |
| --- | --- | --- |
| 101 | c101 | A |
| 101 | c102 | B |
| 102 | c101 | A |

Here:

* StudentID alone → ❌ not unique (a student can take many courses)
* CourseID alone → ❌ not unique (a course can have many students)

But:

* {StudentID, CourseID} together → ✅ unique for each enrollment

👉 So, {StudentID, CourseID} is a **composite key**.

Composite key:

* Candidate keys which were not chosen as primary key

**Foreign key**

### **What is a Foreign Key?**

A **Foreign Key** is a column (or a set of columns) in one table that **references the Primary Key** of **another table**.

🔸 It creates a **relationship** between two tables — like a link between related data.

1. **Students Table**:

| StudentId(PK) | Name |
| --- | --- |
| 101 | Alice |
| 102 | Bob |

2. **Enrollments Table**:

| EnrollmentID | StudentID(FK) | CourseID |
| --- | --- | --- |
| 1 | 101 | C101 |
| 2 | 102 | C102 |

Here:

* StudentID in the Students table is the **Primary Key**.
* StudentID in **Enrollments** table is a **Foreign Key** that refers to the Students table.

👉 This means:

* Each StudentID in the **Enrollments** table **must exist** in the **Students** table.

### **🔒 Why Use a Foreign Key?**

* **Maintains data integrity** (prevents invalid data)
* **Enforces relationships** between tables
* Helps with **joins** in SQL

Database Design: SDLC

Software development life Cycle.

Planning - Analysis - System Design - Building - Testing - Deployment

Purpose of Database design:

Structured organization for efficient data management and retrieval.

Techniques to design Database:

1. Top-down
2. Bottom-up

Steps of Top-down Technique:

* Step 1: Determining Entities
* Step 2: Determining Attributes For Each Entities
* Step 3: Relationships Among Entities
* Step 4: Solving Many to Many Relationships

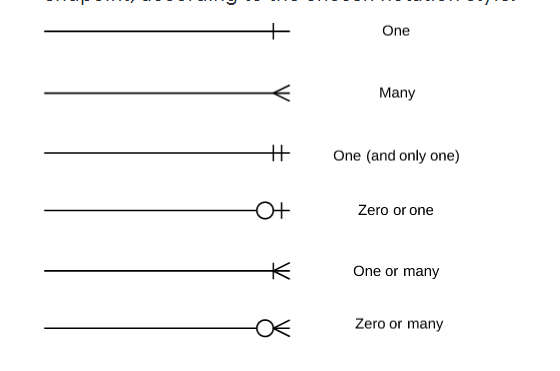
Entity-Relationship (ER) diagram:

An **Entity-Relationship (ER) Diagram** is a **visual representation** of data and their relationships in a database. It shows different entities in a database related to each other through various types of relationships like one-to-one, one-to-many, or many-to-many.

Cardinality:

Relationship cardinality in databases specifies how many instances of one entity are associated with how many instances of another entity.

Tooling for Er Diagram And Creating First Er Diagram:



Anomalies:

Anomalies in databases refer to inconsistencies or unexpected issues that can occur during data manipulation or retrieval.

There are three main types of anomalies:

1. Update anomalies
2. Delete Anomalies
3. Insert Anomalies

Normalization:

Normalization is a database design process that organizes data to minimize redundancy and dependency, resulting in a more efficient, consistent, and manageable database.

Functional Dependency:

Functional dependency in simple terms means that the value of one attribute (or set of attributes) uniquely determines the value of another attribute(s) in a database table.

A **functional dependency** is a relationship between two attributes (columns) in a database table where **one attribute uniquely determines another**.

A → B

This means:

If two rows have the same value of **A**, then they **must have the same value** of **B**.

So, **A functionally determines B**.

| StudentID | Name | DeptID |
| --- | --- | --- |
| 101 | Alice | CSE |
| 102 | Bob | EEE |
| 103 | Alice | CSE |

### **Example Functional Dependencies:**

### StudentID → StudentName ✅ (Because each student ID has exactly one student name)

### StudentID → DeptID ✅ (Each student is in only one department)

### StudentName → StudentID ❌ (Because more than one student can have the same name)

**Normal Forms** (Levels of Normalization)

A series of guidelines that help to ensure that the design of a database is efficient, organized, and free from data anomalies.

Unnormalized Form (UNF):

Data is not organized. Might contain multivalued or repeating fields.

Example:

| StudentID | Name | Course |
| --- | --- | --- |
| 101 | Alice | DBMS, Networking |
| 102 | Bob | DBMS |

**Problem:**

The Course column has **multiple values** (not atomic).

1. 1NF – First Normal Form

Rules:

* Atomic Values(Each field should contain only one value — no lists or sets).
* Unique Column Names(Each column must have a unique name for clarity).
* Positional dependency of data(Data should not depend on the order or position of columns).
* Column should contain data that are of the same type
* Determine Primary key

| StudentID | Name | Course |
| --- | --- | --- |
| 101 | Alice | DBMS |
| 101 | Alice | Networking |
| 102 | Bob | DBMS |

### **✅ Fixes Applied:**

* Course values are **atomic** (only one course per row).
* No repeating groups.
* Data in each column is of the same **type**.
* We can define a **primary key** as (StudentID, Course) — uniquely identifies each row

## **Primary Key in 1NF**

To make the table fully 1NF-compliant:

* Choose a **primary key** to uniquely identify each row.
* In this case:  
   → (StudentID, Course) is a **composite key** → Or you can create a new EnrollmentID column as a surrogate key.

2NF

Rules:

* Must be in 1NF
* Must not contain any non-prime/non-key attribute that is functionally dependent on a proper subset of any candidate key of the relation.(If your table has a **composite key** (a key with 2 or more columns),

then **no non-key column** should depend on just **part** of that key.

It must depend on the **whole key**.)

Proper subset: A set **A** is a **proper subset** of set **B** if:

1. **A is a subset(A set A is a subset of set B if every element of A is also in B) of B**, and
2. **A is NOT equal to B** (A has fewer elements than B)

Problem Scenario: Table in 1NF but not 2NF:

| StudentID | Name | Course |
| --- | --- | --- |
| 101 | Alice | DBMS |
| 101 | Alice | Networking |
| 102 | Bob | DBMS |

### 🔸 Composite Key: (StudentID, Course) — uniquely identifies each row.

Non-prime attribute is Name

🔴 Problem:

* StudentName depends **only on StudentID**, not on the full composite key.
* So this is a **partial dependency** → violates 2NF.

🔹 **Key Terms to Know:**

* **Candidate key (A minimal set of attributes that can uniquely identify a row)**
* **Composite key(A key that is made of more then one column e.g. StudentID + Course)**
* **Non-prime attribute (An attribute that is not part of any candidate key.)**
* **Partial Dependency (A non-prime attribute(not part of the key) depends on part of a composite key, not the whole)**

Solution: Convert to 2NF:

**🔹 Step 1: Break into 2 tables**

**Table 1: Students:**

| StudentID | Name |
| --- | --- |
| 101 | Alice |
| 102 | Bob |

### 

Table 2: Enrollments

| StudentId | Course |
| --- | --- |
| 101 | DBMS |
| 101 | Networking |
| 102 | DBMS |

Now:

* StudentName depends **only** on StudentID (correct place).
* The Enrollment table only has data that depends on the **full composite key**.

Rules of 2NF – In Simple Words

1. Must be in 1NF (Table must have atomic values, no repeating columns)
2. No partial dependency (No non-prime attribute should depend on **part** of a composite key)
3. Every non-prime attribute fully depends on the entire candidate key(Each non-key column must depend on the full key).

3NF

Rules:

* Must be in 2NF
* Must not contain transitive dependency

### **Why is 3NF Needed?**

To remove **transitive dependencies**, which can cause:

* Data redundancy
* Update anomalies
* Inconsistent data

Example (Before 3NF — violates it):

Table: Students

| StudentID | StudentName | DeptID | DeptName |
| --- | --- | --- | --- |
| 101 | Alice | D01 | CSE |
| 102 | Bob | D02 | EEE |
| 103 | Charlie | D01 | CSE |

Analysis:

* Primary key: StudentID
* StudentName and DeptID depend on StudentID → OK
* But DeptName depends on DeptID, not directly on StudentID

This is called a **transitive dependency**.

**Transitive dependency**:{

If this is true x→Y & Y → z

Then must it true x→z

}

StudentID → DeptID → DeptName

This violates 3NF.

Fix (Bring to 3NF):

1. Students Table

| StudentID | StudentName | DeptID |
| --- | --- | --- |
| 101 | Alice | D01 |
| 102 | Bob | D02 |
| 103 | Charlie | D01 |

2. Departments Table

| DeptID | DeptName |
| --- | --- |
| D01 | CSE |
| D02 | EEE |

Now:

* DeptName depends only on DeptID
* No transitive dependency
* ✅ Now the design is in **3NF**