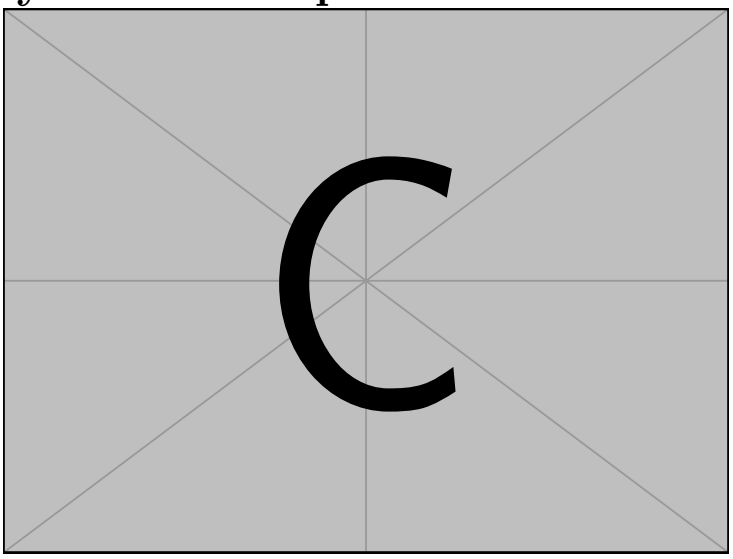
DBMS Notes: Unit 2 The

Entity-Relationship & Relational Models



Course Code: CS502
Generated: October 8, 2025

Contents

1 The Entity-Relationship (ER) Model

The ER model is a high-level, conceptual data model used to design databases. It provides a graphical "blueprint" of the database structure that is easy for both technical and non-technical stakeholders to understand.

1.1 Core Components of an ER Diagram

Entity: A real-world object or concept. Represented by a **rectangle**. E.g., 'Student', 'Course'.

Attribute: A property or characteristic of an entity. Represented by an **ellipse**. E.g., 'studentID', 'courseName'.

Relationship: An association between two or more entities. Represented by a **diamond**. E.g., a 'Student' *enrolls in* a 'Course'.

1.1.1 Types of Attributes

- **Simple vs. Composite:** A composite attribute can be broken down (e.g., 'Address' → 'Street', 'City', 'Zip'), while a simple one cannot (e.g., 'Age').
- **Single-valued vs. Multi-valued:** A multi-valued attribute can have more than one value (e.g., 'PhoneNumber'), represented by a double ellipse.
- **Stored vs. Derived:** A derived attribute's value is calculated from another attribute (e.g., 'Age' can be derived from 'DateOfBirth'). Represented by a dashed ellipse.
- **Key Attribute:** An attribute that uniquely identifies an entity instance. The text is underlined.

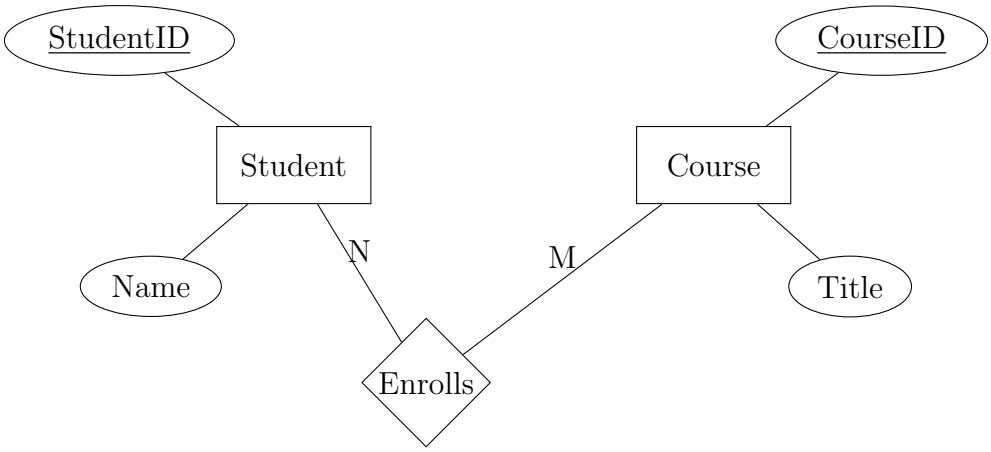
1.2 Cardinality Constraints

Cardinality specifies the number of instances of one entity that can be associated with instances of another entity.

- **One-to-One (1:1):** E.g., A 'Country' has one 'Capital City'.
- **One-to-Many (1:N):** E.g., One 'Professor' teaches many 'Courses'.
- **Many-to-Many (M:N):** E.g., Many 'Students' can enroll in many 'Courses'.

1.3 Example ER Diagram

A simple diagram for a university system.



2 The Relational Model

The relational model, proposed by E.F. Codd, is the most widely used logical data model. It represents the database as a collection of tables, called **relations**.

2.1 Terminology

- **Relation:** A table with columns and rows.
- **Attribute:** A named column of a relation.
- **Tuple:** A row in a relation.
- **Domain:** The set of all possible values for an attribute. E.g., the domain for ‘Age‘ might be integers from 0 to 120.
- **Schema vs. Instance:** The schema is the design of the relation (table name and columns), while the instance is the actual data in the relation at a specific moment.

2.2 Keys

Keys are crucial for identifying tuples and linking relations.

- **Super Key:** A set of one or more attributes that can uniquely identify a tuple.
- **Candidate Key:** A minimal super key (no attribute can be removed).
- **Primary Key:** One candidate key chosen by the DBA to be the main identifier for a relation. It cannot contain ‘NULL‘ values.
- **Foreign Key:** An attribute (or set of attributes) in one relation that refers to the primary key of another relation. This is how relationships are formed.

2.3 Mapping ER Diagrams to Relational Schemas

There is a well-defined algorithm for converting an ER design into relational tables:

1. **Entities → Tables:** Each strong entity becomes a table. The entity's attributes become the table's columns.
2. **Multi-valued Attributes → New Table:** A multi-valued attribute requires its own separate table, including a foreign key pointing back to the parent entity's table.
3. **Relationships → Foreign Keys or New Tables:**
   - For a **1:N** relationship, add a foreign key to the table on the "N" side that references the primary key of the "1" side.
   - For a **M:N** relationship, create a new "junction" or "bridge" table. This table's primary key is composed of the foreign keys from both related entities.

3 Smart Quirks & Key Insights

3.1 ER Diagrams are Blueprints, not Photographs

An ER diagram is a high-level conceptual tool for modeling business rules. It is meant to be a discussion point between designers, developers, and clients. It captures *what* needs to be stored, not *how*. The process of converting it to a relational model (and then implementing it in SQL) is where the specific, logical details are hammered out.

3.2 Foreign Keys are the "Glue" of the Database

Primary keys give records an identity, but **foreign keys give the database its structure**. They enforce **referential integrity**, which prevents "orphan" records. For example, you cannot create an 'Order' for a 'CustomerID' that doesn't exist. This single mechanism is responsible for maintaining consistency across the entire database, a task that would be a nightmare to manage manually in application code.

3.3 Weak Entities are "Moochers"

A weak entity is one that cannot be uniquely identified by its own attributes alone; it depends on the entity it's associated with (the "strong" entity).

- **Example:** A 'Room' in a 'Building'. The room number '101' is not unique on its own; it's only unique within a specific building.
- **Quirk:** The primary key of a weak entity's table is a composite key, formed by combining the primary key of the strong entity (as a foreign key) and the weak entity's own partial key (its "discriminator"). So, the PK for 'Room' would be '(BuildingID, RoomNumber)'.