CSCI3170 Introduction to Database Systems

Ryan Chan

October 11, 2025

Abstract

This is a note for **CSCI3170 Introduction to Database Systems**.
Contents are adapted from the lecture notes of CSCI3170, prepared by Michael Ruisi Yu, as well as some online resources.
This note is intended solely as a study aid. While I have done my best to ensure the accuracy of the content, I do not take responsibility for any errors or inaccuracies that may be present. Please use the material thoughtfully and at your own discretion.
If you believe any part of this content infringes on copyright, feel free to contact me, and I will address it promptly.
Mistakes might be found. So please feel free to point out any mistakes.

Contents

Chapter 1

Introduction

1.1 Overview

Data will always need to be stored, manipulated, accessed, shared, and transmitted. Thus, we require certain methods to handle it.

A data table (or data frame) is a two-dimensional structure. Regarding the data itself, we generally categorize it into three types. The first is **Unstructured Data**, which refers to information that does not follow a specific format, such as the statement: "a university has 10,000 students." Next, we have **Semi-structured Data**, which has some organizational elements (e.g., tags, hierarchies) but is still difficult to process directly. Finally, we have **Structured Data**, the most organized type, stored in predefined formats such as tables with rows and columns.

To manage such vast amounts of data, we use **Database Management Systems (DBMS)**, which are software packages designed to maintain and utilize large collections of data.

By using a DBMS to store data, we ensure data independence, data integrity, security, concurrent access, and crash recovery.

First, we begin with the **conceptual model**.

1.2 Entity Relationship Model

By receiving a set of requirements, we need to design an application. For the data behind it, we must create a database schema that provides a logical view of the data model. However, the main challenge is how to build a database for this application that meets all the requirements — in other words, how to construct a systematic database.

In this case, we use Chen's **Entity-Relationship (ER) Model**. In this model, there are two major components: **Entities** and **Relationships**. An **Entity** is a collection of attributes that describe an object of interest, while a **Relationship** represents the association between entities (objects). There is also a third component, the **Attribute**, which is a data item describing a property of interest.

Let us now look at the details.

1.2.1 Entity

An **Entity** is something that is distinguishable from other objects. For example, entities can be *Classrooms*, *Students*, etc. Entities represent things in the real world, and **Attributes** describe their properties.

Some attributes are simple (also called *atomic*), meaning they cannot be split into simpler components. Each simple attribute of an entity type holds one value. It is associated with a **value set** (or *domain*), which specifies the possible values that may be assigned to that attribute for each individual entity.

An entity is described using a set of attributes whose values distinguish one entity from another of the same type. An **Entity Set** is a collection of such entities (instances of the entity type).

For each attribute associated with an entity set, we must identify a domain of possible values.

To describe data in terms of a data model, we use a **schema**. For example, for a *Student* entity set, we may define the schema:

Students(sid: string, name: string, login: string, age: integer, gpa: real)

A good entity schema should include attributes that are meaningful, well-defined, and capable of being filled with valid values.

1.2.2 E-R Diagrams

Data modeling is typically divided into three tiers:

1. **Semantic (High-level or Conceptual)** — provides an initial description of the data in the enterprise.

2. **Implementation (or Record-based)** — describes how data can be organized using models such as the Relational, Network, or Hierarchical model.

3. **Low-level (or Physical)** — specifies details such as record formats, access paths, and storage structures.

To develop a database, we must first analyze the requirements, then design and implement the data model. The **Entity-Relationship (E-R) diagram** is often used for conceptual modeling.

For example, in Figure 1.1, the E-R diagram represents the following description:

> "There is one strong entity type called *Car*. It has a multivalued attribute to describe its color. It also has a composite attribute *Registration*, which consists of *Registration Number* and *State*. Additionally, it includes several other attributes such as *Make*, *Model*, and *Year*."

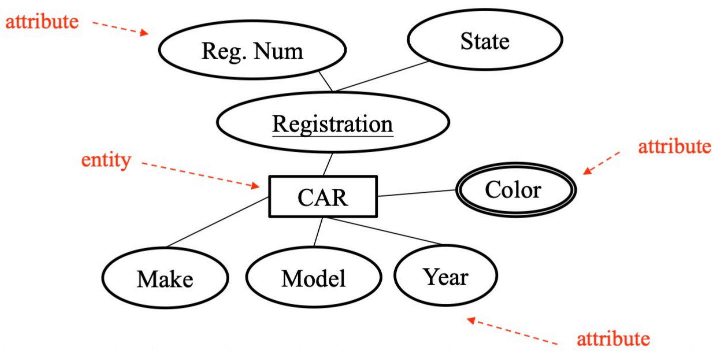


Figure 1.1: E-R Diagram: Car entity

1.2.3 Relationship

We have discussed **Entities** and **Attributes** in the Entity–Relationship (E-R) Model. The second major component is the **Relationship**.

A relationship can also have **descriptive attributes** (see Figure 1.2). Descriptive attributes are used to record information about the relationship itself, rather than about any of the participating entities.

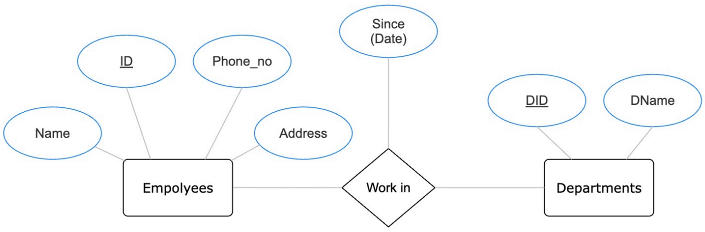


Figure 1.2: E-R Diagram: Relationship

Relationships represent logical links between two or more entities. Any set of entities is not limited to participating in only one relationship with each other.

An **Entity–Occurrence Diagram** shows the relationships between individual occurrences (instances) of particular entities.

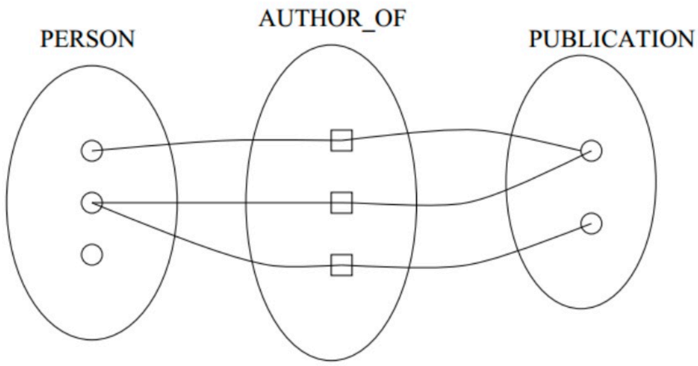


Figure 1.3: Many-to-Many Relationship

There are three main types of relationships. To distinguish among them, we apply two types of constraints: **cardinality** and **participation**.

1.2.4 Cardinality

The **cardinality constraint** (or *cardinality ratio*) specifies the number of relationship instances an entity can participate in. The three common types are:

- **1:1 (one-to-one)** — each entity instance is associated with at most one instance of another entity.
- **1:N (one-to-many)** — one entity instance can be associated with many instances of another entity.
- **M:N (many-to-many)** — multiple instances of one entity can be associated with multiple instances of another entity.

We can use standard **entity–occurrence diagrams** to visualize such relationships.

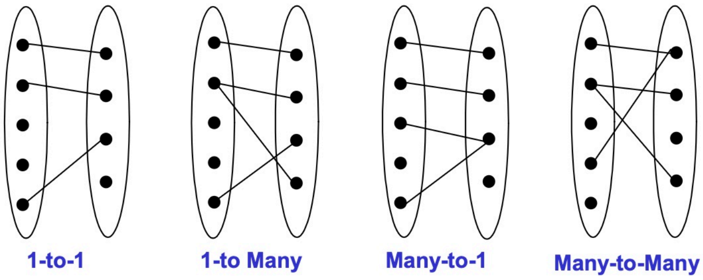


Figure 1.4: Standard Entity-Occurrence Diagrams

One-to-Many

One-to-many constraint from A to B: an entity in B can be associated with at most one entity in A.

To represent such relationships, we use arrows in the diagram. An arrow indicates that one entity can be uniquely associated with a relationship instance.

For example, in the following case, each *Employee* belongs to one *Department*, while a *Department* may have multiple employees. We use an arrow pointing from the *Employee* entity set to the *works_in* relationship to represent this constraint.

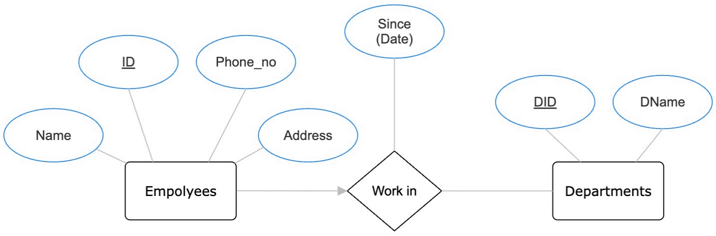


Figure 1.5: E-R Diagram: One-to-many

One-to-One

If a relationship between A and B satisfies the one-to-one mapping constraint, i.e., each entity in A is related to at most one entity in B, and each entity in B is related to at most one entity in A,

we represent this relationship with two arrows, one pointing from A to B and one from B to A.

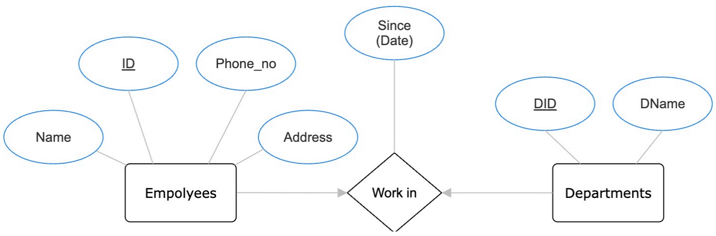


Figure 1.6: E-R Diagram: One-to-One

Many-to-Many

If an entity in A can be associated with any number of entities in B, and an entity in B can be associated with any number of entities in A, this indicates that there is no restriction on the mapping.

1.2.5 Participation

In a one-to-one relationship, as shown in Figure 1.6, if we change the *works_ in* relationship to *manages*, one question arises: is there at least one manager in each department? The cardinality constraint does not provide this information. Therefore, we need **participation constraints**.

We can classify participation in relationships as follows:

- **Total participation** — every entity in the entity set must participate in at least one relationship.
- **Partial participation** — an entity in the entity set may not participate in any relationship.

In an E-R diagram, total participation is represented by a double line, while partial participation is represented by a single line.

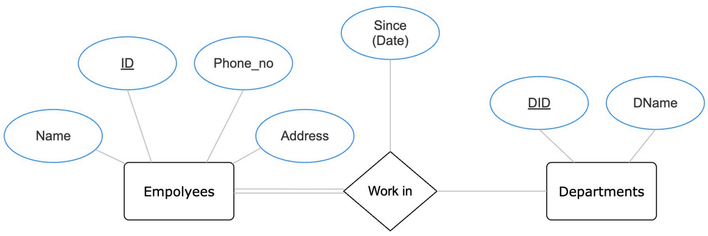


Figure 1.7: E-R Diagram: Total Participation

As shown in Figure 1.7, this indicates that every *Employee* must work for some *Department*.

1.2.6 Attributes

The final component of Entity–Relationship (E-R) modeling is **Attributes**.

Some semantics cannot be captured using simple (atomic) attributes. In such cases, we use **multivalued attributes**, which are represented by a double ellipse in an E-R diagram.

We also have **composite attributes**. Unlike simple attributes, which are indivisible, composite attributes can be divided into smaller subparts, each representing a more basic attribute with independent meaning.

Another type is **derived attributes**. These are attributes whose values can be derived from other attributes, rather than being stored directly. For example, a person's *age* can be derived from their *date of birth*. In an E-R diagram, derived attributes are represented by a dashed ellipse.

1.3 Key

In the discussion of entities, we mentioned that an entity represents a set of objects with the same attributes within a data model. How, then, can we distinguish different entities? The answer is through **keys**.

1.3.1 Overview

A key may contain more than one attribute. In some cases, we need to add attribute(s) as a key. In an E-R diagram, keys are represented by *underlined attributes*.

A **natural key** is a column or a set of columns that already exist in a table and uniquely identify a record in that table.

A **super-key** is any set of attributes that can uniquely identify an entity. If a key consists of more than one attribute, it is called a **composite key** or **compound key**.

A **candidate key** is a minimal set of attributes whose values uniquely identify an entity in the entity set. As there could be more than one candidate key, **primary key** is a candidate key chosen to serve as the main key for the entity set.

For example, consider the following schema:

Lecture(lecturer: string, course_code: string, location: string, date: date, time: string)

In this schema, {location, date, time} is a key, while {lecturer, location, date, time} is not minimal, but it is a **super-key**.

However, it is possible that no existing attribute can uniquely identify an entity. In such cases, we can provide a **surrogate key**, which is a system-generated value used to uniquely identify a record in a table.

All entity sets have a **primary key**. They are considered independent if they possess a primary key.

For entity sets that do not have a primary key, i.e., they are dependent on another entity set, they are called **weak entity sets**. In an E-R diagram, a weak entity is represented by a double rectangle.

We can make a weak entity stronger by providing a **surrogate key**. However, its existence still depends on the existence of an **identifying entity set**. A weak entity must be related to the identifying entity set via a total one-to-many relationship from the identifying entity set to the weak entity set.

A weak entity also has a **partial (discriminator) key**, which uniquely identifies weak entities only within the context of the identifying entity.

Note that in an E-R diagram, the partial key of a weak entity set is denoted with a *dashed underline*, and the identifying relationship is represented by a *double diamond*.

1.3.2 Keys in Relationships

A relationship must be uniquely identifiable. The participating entity instances contribute to the identification of each relationship instance.

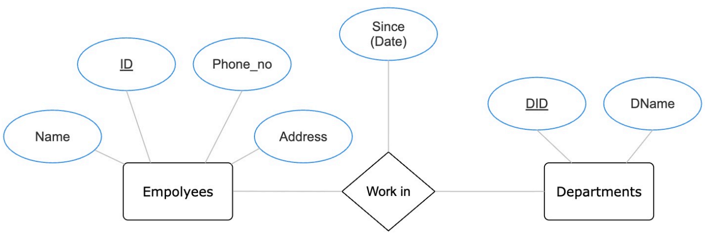


Figure 1.8: E-R Diagram

In Figure 1.8, each relationship is uniquely identified by the combination of the *Employee* ID and the *Department* DID. That is, for each pair, there cannot be more than one corresponding relationship instance.

The concept of keys is also used to identify relationships, similar to entities.

- For a relationship among $E_1, \ldots, E_k$ with no mapping constraint (many-to-many), the primary key is normally the union of the primary keys of $E_1, \ldots, E_k$.
- For a one-to-many relationship, an entity set E has a key constraint in a relationship set R, such that each entity in E participates in at most one relationship in R. Hence, an entity in E can uniquely identify a relationship in R, and the key of E can be used as the key in R.
- For a one-to-one relationship between two entity sets E and F, both key(E) and key(F) can serve as keys for the relationship set.

1.4 Conceptual Design

We say a relationship is **binary** if it links two entities.

There are also relationships of higher degree. A **ternary relationship** has degree 3. Sometimes a relationship might involve two entity instances from the same entity type; this is called a **recursive relationship**. For example, an *Employee* can be the supervisor of another employee, where this relationship links two entities within the *Employees* entity set.

For a **non-binary relationship**, suppose $n \geq 2$ for $E_1, E_2, \ldots, E_n$. It cannot always be replaced by multiple binary relationship sets. Consider the following example: the existence of (s, p), (j, p), and (s, j) does not imply the existence of (s, j, p). This phenomenon is known as the **connection trap**.

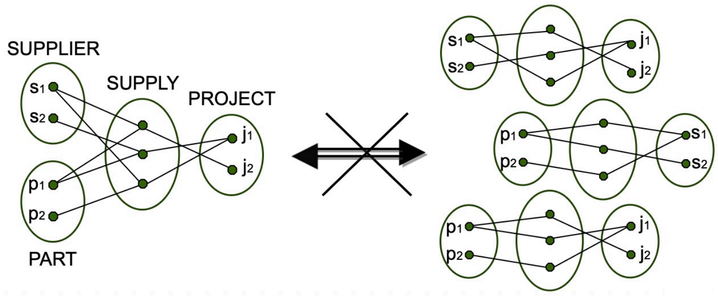


Figure 1.9: Connection Trap

In the conceptual design phase, we need to consider whether a concept should be modeled as an **attribute** or an **entity**, and whether it should be modeled as an **entity** or a **relationship**. We also need to decide whether to use **binary** or **ternary relationships**.

Sometimes, it is necessary to model a relationship between a collection of entities and relationships. **Aggregation** allows us to indicate that a relationship set (represented by a dashed box) participates in another relationship set.

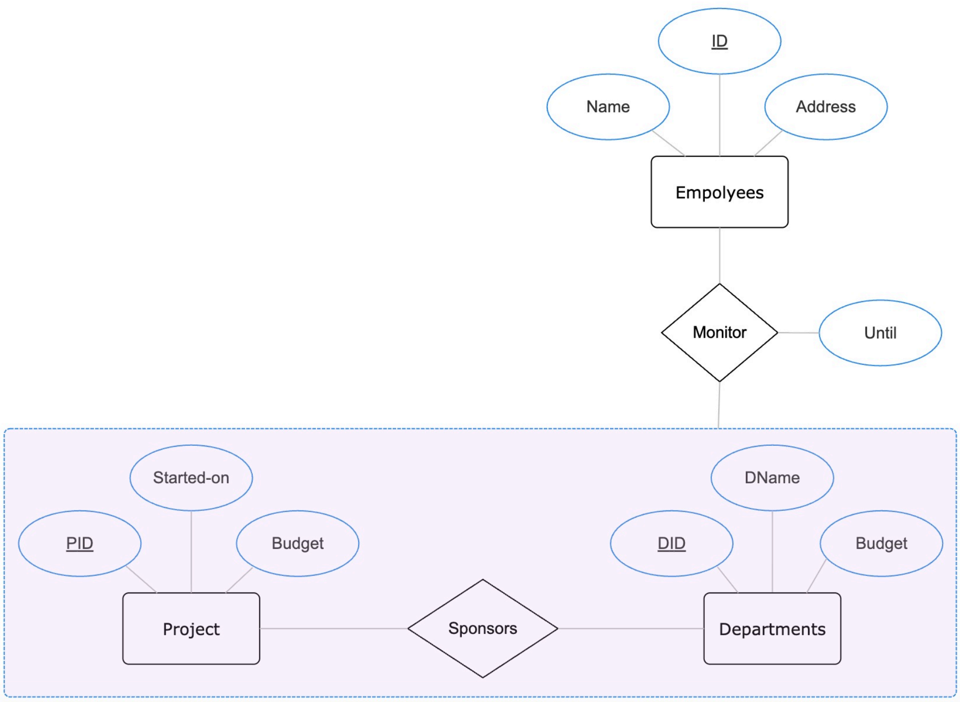


Figure 1.10: Aggregation

For example, in Figure 1.10, *Monitor* should be modeled as a relationship set that associates the *Sponsors* relationship (rather than the *Project* or *Departments* entities) with the *Employees* entity.

1.5 Class Hierarchies

It is natural to classify the entities in an entity set into **subclasses**, because we may want to add descriptive attributes that are meaningful only for entities in a subclass, and we may want to identify which entities participate in certain relationships.

A **class hierarchy** can be viewed in two ways:

- A class is **specialized** into subclasses.
- Subclasses are **generalized** by a super-class.

For example, in *Employees*, there could be *Hourly Employees* and *Contract Employees.* Note that attributes of the super-class are inherited by entities in the subclass.

We can specify two kinds of constraints with respect to ISA hierarchies:

1. **Overlap constraints**: Determine whether subclasses are allowed to contain the same entity.

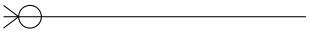
Example: Can an employee belong to both *Hourly Employees* and *Contract Employees*?

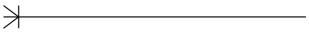
2. **Covering constraints**: Determine whether the entities in the subclasses collectively include all entities in the super-class.

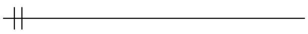
Example: Does every *Employee* entity also have to be either an *hourly employee* or a *contract employee* entity?

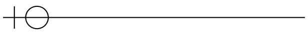
1.6 Crows Feet Notation

1.6.1 Relationships

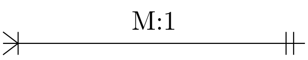

Zero or More


One or More

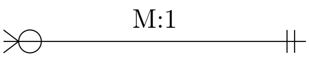

One and only One


Zero or One

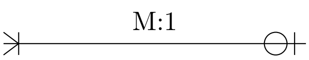
1.6.2 Many-to-One



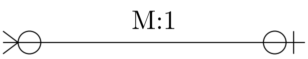
a one through many notation on one side of a relationship and a one and only one on the other



a zero through many notation on one side of a relationship and a one and only one on the other



a one through many notation on one side of a relationship and a zero or one notation on the other



a zero through many notation on one side of a relationship and a zero or one notation on the other

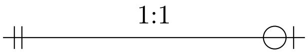
1.6.3 Many-to-Many

a zero through many on both sides of a relationship

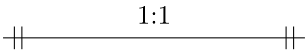
a one through many on both sides of a relationship

a zero through many on one side and a one through many on the other

1.6.4 One-to-One



a one and only one notation on one side of a relationship and a zero or one on the other



a one and only one notation on both sides

Chapter 2

Logical Mapping

We have discussed the E-R diagram, which represents the **conceptual schema**. Once we have the E-R diagram, we can derive a **relational schema**.

2.1 Relational Model

Let us first take a look at the **relational model**.

2.1.1 Relation

A **relation** consists of a **relation schema** and its **relation instance**. One can think of a relation as an *entity set*, where the relation instance is the set of records that populate the relation and corresponds to the entities in that set.

A **relation schema** describes the column headers for the table. It specifies the relation's name, the name of each field, and, if detailed, the domain of each field. For example, we can have:

Students(sid: string, name: string, login: string, age: integer, gpa: real)

More formally, a relation schema R is a finite set of attribute names:

$$R = (A_1, A_2, \dots, A_n)$$

For each attribute name A_i, there is a corresponding **domain** D_i, which is the set of values that A_i can take. Attribute values are required to be **atomic** (indivisible). The value `NULL` is also considered a member of every domain.

A relation r on the relation schema R is a subset of $D = D_1 \times D_2 \times \cdots \times D_n$. Thus, a relation is a set of n-tuples $(a_1, a_2, \dots, a_n)$ where each $a_i \in D_i$. The current values (instance) of a relation are specified by a table. An element t of r is called a **tuple** and is represented by a row in the table.

The relational model requires that no two rows be identical. Thus, a relation is defined as a set of unique tuples (rows). This implies that the order in which the rows are listed, as well as the order of the fields, is not important.

> **Remark.** An unordered collection of elements is called a *set*, while an ordered collection of elements is called a *list*.

In the relational model, the concept of **keys** remains fundamental.