DATABASE MANAGEMENT SYSTEM

DBMS Core Group

Authors	Gaurav Kansal, B P Sharma, Pratibha Singh, Nidhi Singh, Anand Kr. Srivastava, Ritin Behl, Sachin Goel
Authorized by	
Creation/Revision Date	April 2021
Version	1.0

UNIT -2
ER MODEL

Table of Contents

1. Introduction

Software applications are created to manage data and to help transform data into information. A database design is an essential component of any application development that supports and automates business processes. Database designing is a necessary step in the process of creating any complex software. It helps developers understand the domain and organize their work accordingly. Developing a concise and extensible database design is critical for arriving at the right foundation plan while building a house.

The overall design of the database of an organization is called a **schema**. We will explain here different phases in the overall design of the database. These phases are mainly divided based on different levels of abstraction. The schemas defined at the design and implementation levels are:

1. Conceptual schema
2. Logical schema
3. Physical schema
4. External level schema/View level schema/Subschema

The **conceptual schema design** is a high-level data model, providing a conceptual framework to specify the data requirements of the database users and how the database will be structured to fulfill these requirements. The database designer needs to interact extensively with domain experts and users to carry out this task. The schema developed at the conceptual design phase provides a detailed overview of the enterprise.

A **conceptual schema** or **conceptual data model** describes the things of significance to an organization (*entity types*), about which it is inclined to collect information, and its characteristics (*attributes*), and the associations between pairs of those things of significance (*relationships*).

The **logical schema** defines the design of the database at the conceptual level of the data abstraction. In this step, the conceptual schema is transformed from the high-level data model into the implementation data model. It defines how the database should be implemented using a specific DBMS. This schema defines all the logical constraints that need to be applied to the data stored in the database.

The **physical schema** is the database design at the physical level of data abstraction. The logical schema is mapped to the physical schema using RDMBS tools like Microsoft SQL Server, Oracle SQL, or IBM's DB2. It describes how the data is organized in files, internal storage structure, and access paths.

View Schema or external schema, also called **subschema**, defines the design of the database at the view level of the data abstraction. It describes how an end-user will interact with the database system. There are many view schema for a database system. Each view schema defines the view of data for a particular group of people. It shows only those data to a view group in which they are interested and hides the remaining details.

In the database design phases, data is represented by using a specific data model. The data model is a collection of concepts or notations for describing data, data relationships, data semantics, and data constraints. Most data models also include a set of basic operations for manipulating data in the database.

Data modeling is a technique to define and organize the data requirements of an enterprise. It allows creating a visual description of the business by analyzing, understanding, and clarifying its data requirements. This chapter introduces Data Modelling, its development, and concepts around it.

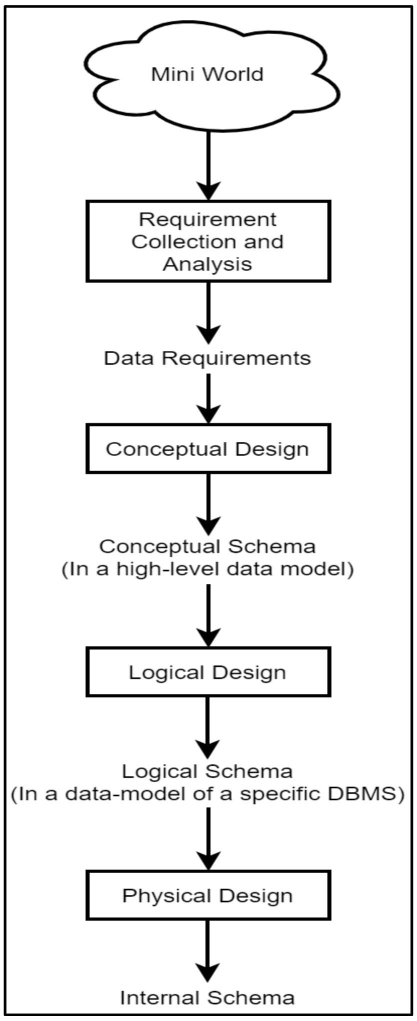

Mini World
Requirement Collection and Analysis
Data Requirements
Conceptual Design
Conceptual Schema (In a high-level data model)
Logical Design
Logical Schema (In a data-model of a specific DBMS)
Physical Design
Internal Schema

Figure 2.1 Representation of Data Modelling Process

2. Case Study

<u>Case Study – College Management System</u>

The primary purpose of this case study is to design a database to construct a student information management system in the department of student affairs – ABESEC to transform the work in this department from manually to a computer-based system, which leads to providing accuracy, efficiency, security, and so on.

2.1 User Requirement

After talking with the college employee who is responsible for the administration of the department of Student Affairs at the college, it has been determined the required information in the system. This determination is according to the actual needs of the college. The required information for the student information management system is described as follows:

- Student information
- Faculty Information
- Course information
- Other information

The employee has shared the below-mentioned requirement story.

In a College, there are several departments, and each department has one head of the department (HOD). Department has a name, its location, and students that belong to the department. A student can belong to only one department, and a department can have many students. If a department has recently come into existence, it might not have any students. Students have roll number, name, date of birth, gender, hobby, phone number, and address. Faculty has a name, designation, date of joining, gender & salary and belongs to a particular department. A department can run many courses with assigned credits, and students can study any number of courses being offered by various departments. There are sections within each department, and each section has many students. Each section has its name, maximum capacity of that section, and the number of students in that section. Students need to do one mini-project individually. Each available mini-project has to be chosen based on its name, domain, subject, and description. Faculty members can teach multiple courses in multiple departments. One course can be taught by many faculty members across departments. Faculty members can be part of multiple research projects. These projects are either sponsored by the government, industry, or the college itself. The faculty member can do one or more projects, and one project can have more than one faculty member. Research projects have a fixed duration, and their status needs to be tracked regularly.

After collecting the user requirement, just go through the entire story again and then break the requirement story into well-defined parts to understand the requirements in a better way.

In a College,

- There are several departments, and each department has many faculties.
- Out of the faculties from each department, one faculty is acting as Head of Department.
- Department has a name, Id, phone extension, specific mailing address, and Students that belong to the department.
- Students can belong to only one department at a time.
- Department can have more than one or no student.
- Students have their name, unique identification number, address, age, gender, hobbies, and other information.
- Faculty also have information similar to Students except for hobbies.
- A student studies different Courses.
- A department can run many courses, and a course can run in many departments.
- Faculty teaches these Courses. Faculty can teach more than one course.
- Department can run many sections.
- Many students can be in one section, and each section has its name and the max capacity.
- A student must do one mini-project, and one mini-project can be opted by one student only.
- Faculty members can teach in multiple Departments.
- Each course can be taught by many faculty members or no one.
- Faculty members are also working on multiple research projects.
- Research projects are either sponsored by the government, industry, or the college itself.
- One project can have more than one faculty member, and one faculty member can work on more than one project.

After understanding the requirements, we will start designing the database. ER Diagram is the first step of designing any database. Now we will understand the concept of ER Model.

2.2 Basic Concept of ER Model (Entity-Relationship Model)

2.2.1 ER Modelling

The entity-relationship (ER) data model is based on a perception of a real-world that consists of a collection of basic objects, called **entities**, and of **relationships** among these objects. An entity is a “thing” or “object” in the real world distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities.

A set of **attributes** describes entities. For example, the attributes account-number and balance may describe one particular account in a bank, and they form attributes of the account entity. Similarly, attributes customer-name, customer-street address, and customer-city may describe a customer entity.
The entity-relationship (ER) data model describes data as **entities, attributes and relationships**. It was developed to facilitate database design by allowing the specification of an enterprise schema, representing the overall logical structure of a database.
Here we will try to understand key concepts of ER Modelling and how to use them for database design using a set of graphical symbols.
The database design created using these graphical symbols is called Entity-Relationship Diagram or ER Diagram or simply ERD.

2.2.2 Entities, Attributes and Relationships

Entity

An entity is a real-world object which can be a person, place, or thing that can be uniquely identified and distinguished from other objects. For example, every **STUDENT** has a unique roll number. Every **DEPARTMENT** has a unique department code. Every **COURSE** has a unique course code Every **COUNTRY** has unique country code, and every **CAR** has a unique registration number. Here, **STUDENT, DEPARTEMENT, COURSE, COUNTRY,** and **CAR** are few examples of entities.

If we cannot distinguish an item from another item, it is simply an object but not an entity. E.g., each leaf on a tree is an object but not an entity since we cannot uniquely identify every leaf on a tree.

Entities can be tangible or intangible. Tangible entities exist in the real world physically, e.g., STUDENT, CAR, etc. Intangible entities exist only logically and have no physical existence, e.g., COURSE, DEPARTMENT, etc.
Entities are represented in a rectangular box in an entity-relationship diagram.

Figure 2.2 Representation of an Entity

Attributes

Every entity has a set of properties that describes the entity. For example, a student can have attributes like roll number, first name, middle name, last name, date of birth, gender. A department can have attributes like dept id, dept name, and CAR can have attributes like registration number, make, model, color, price, etc.

Attributes are represented in oval shape attached to the entity.

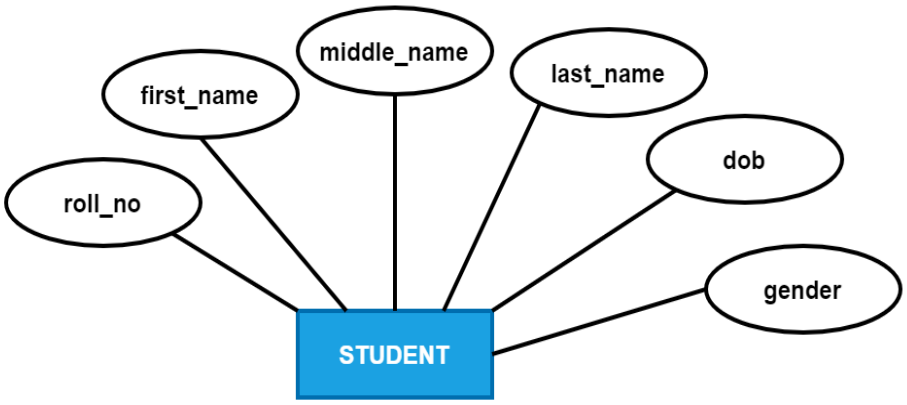


Figure 2.3 Representation of Entity with attributes

Entity Type

An **entity type** defines the set of entities having a common set of attributes, e.g., all the students in a college or university will have the same set of attributes, then STUDENT becomes entity type.

Each entity type is defined by its name and attributes, as shown in the above image.

Entity Set

The collection of entities of a particular entity type in the database is called an **entity set** at any point in time. It is also called the extension of the entity type. The entity set is usually referred to using the same name as the entity type.

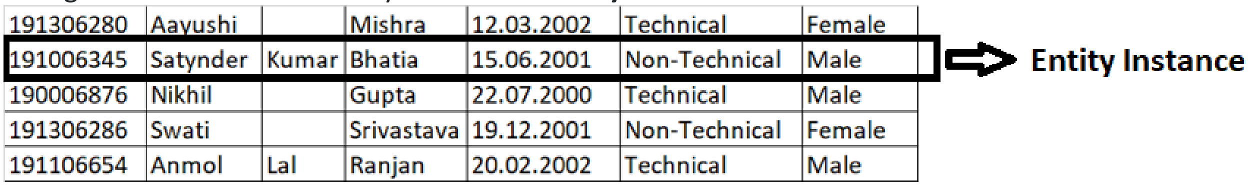
Example:
Entity Type: STUDENT
Entity Set:

191306280	Aayushi		Mishra	12.03.2002	Technical	Female
191006345	Satynder	Kumar	Bhatia	15.06.2001	Non-Technical	Male
190006876	Nikhil		Gupta	22.07.2000	Technical	Male
191306286	Swati		Srivastava	19.12.2001	Non-Technical	Female
191106654	Anmol	Lal	Ranjan	20.02.2002	Technical	Male

Figure 2.4 Representation of an entity set

Entity Instance

A single occurrence of an entity is called an **entity instance**.



191306280	Aayushi		Mishra	12.03.2002	Technical	Female
191006345	Satynder	Kumar	Bhatia	15.06.2001	Non-Technical	Male
190006876	Nikhil		Gupta	22.07.2000	Technical	Male
191306286	Swati		Srivastava	19.12.2001	Non-Technical	Female
191106654	Anmol	Lal	Ranjan	20.02.2002	Technical	Male

Entity Instance

Figure 2.5 Representation of an entity Instance

Types of Attributes

The attributes of an entity can be different types

- General Attribute or Attribute
- Key Attribute
- Multivalued attribute
- Composite Attribute
- Derived Attribute

Attributes

All the characteristics or properties defined for an entity are it’s attributes, e.g., the STUDENT entity type can have the attributes like roll_no, first_name, middle_name, last_name, dob, gender, etc.
Use **Oval** to represent an attribute.

Key Attributes

The attributes which help in the unique identification of entities are called key attributes. Such attributes have unique values, e.g., roll_no of a STUDENT, registration_no of a CAR, course_id of a COURSE, and dept_id of a DEPARTMENT are the key attributes.
While representing such attributes, we need to **underline** them.

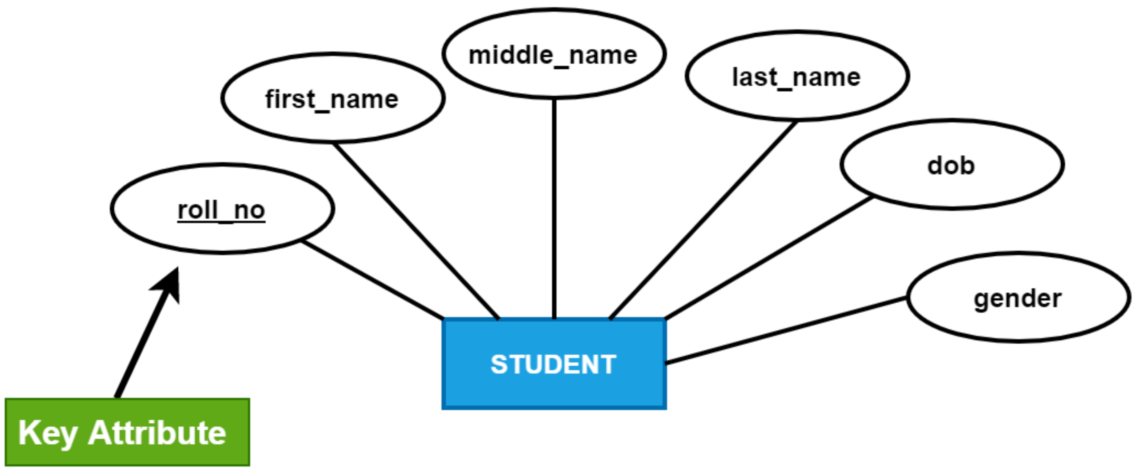


Figure 2.6 Representation of Key Attribute (roll_no)

Multivalued attribute

The attribute which can have multiple values for an entiry type is called a multivalued attribute, e.g., a STUDENT can have multiple phone numbers using attribute phone_no.
Use **double line oval** to represent the multi-values attribute.

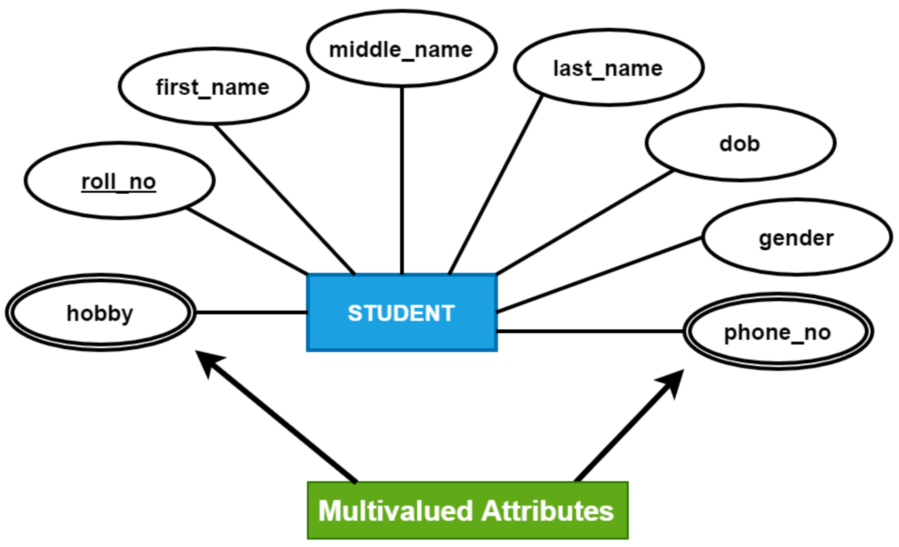


Figure 2.7 Representation of Multivalued Attribute (phone_no)

Composite Attribute

An attribute composed of multiple sub-attributes is called a **composite attribute**, e.g., the address attribute can be divided into five sub-attributes house_no, street_name, city, state, pin_code.

Use **rounded brackets** to represent such attributes inside the oval shape.

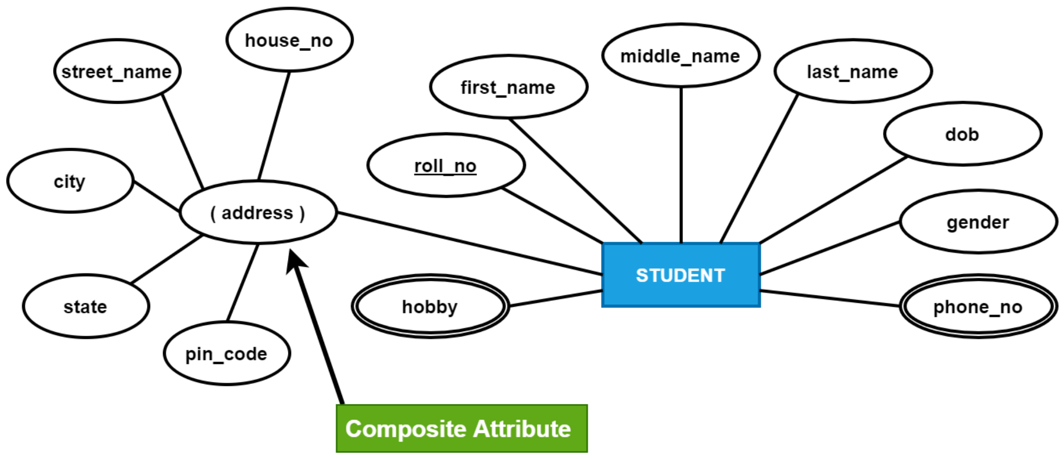


Figure 2.8 Representation of Composite Attribute (address)

Derived Attribute

An attribute whose value is calculated from another attribute(s) is called a **derived attribute**, e.g., the **age** attribute can also be derived from another attribute, **dob** (date of birth).
Use **dotted line oval** shape to represent a derived attribute.

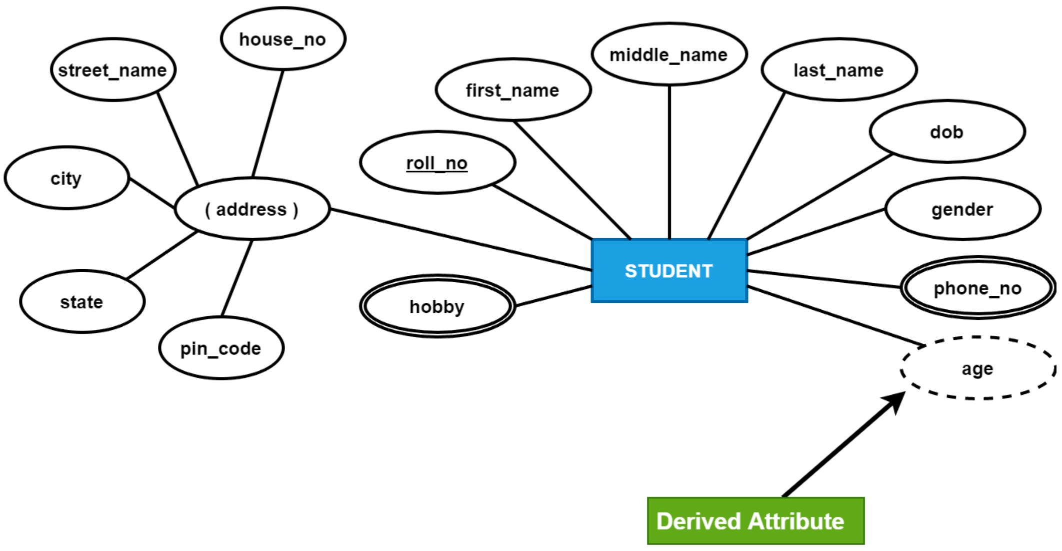


Figure 2.9 Representation of Derived Attribute (age)

Different Entity Types

Entity Type can be of two types:

- Regular (Strong) Entity Type
- Weak Entity Type

Strong Entity Type:

Strong entity types are those entity types that have a key attribute(s).

A rectangle in the ER diagram represents a strong entiry type.

In the given example, roll_no identifies each student uniquely, and hence, we can say that STUDENT is a strong entity type.

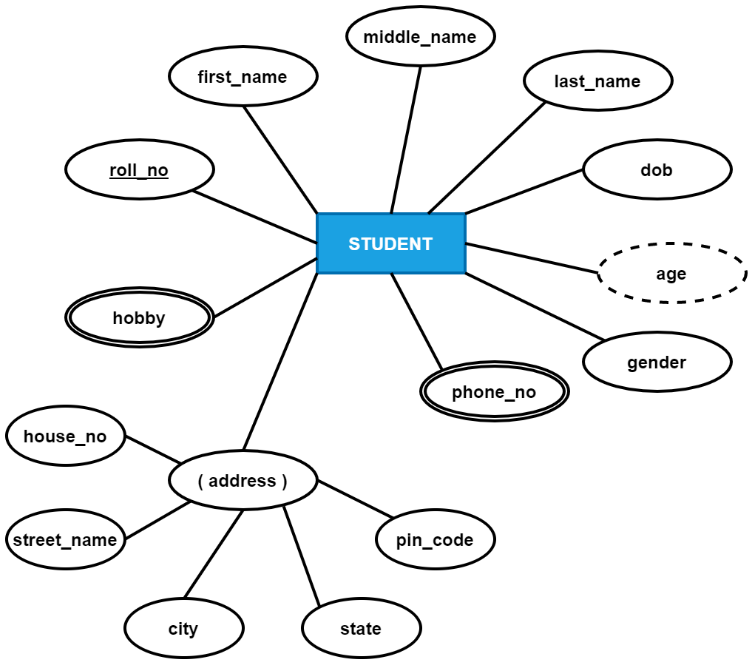


Figure 2.10 Representation of Strong Entity (STUDENT)

Weak Entity Type

An entity type may not have a key attribute(s) to identify each entity instance uniquely. Such an entity type is termed a weak entity type. For a weak entity type to be meaningful, it must be associated with another entity type, called the identifying or owner entity type. The weak entity type is said to be existence dependent on the identifying (owner) entity type. The identifying entity type is said to own the weak entity type that it identifies.

The relationship associating the weak entity type with the identifying entity type is called an identifying relationship.

Although a weak entity type does not have a primary key, it normally has a partial key, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. In the worst case, a composite attribute of all the weak entity's attributes will be the partial key. The partial key is sometimes called the discriminator. For example, we have a weak entity type SECTION and the identifying (owner) entity type is DEPARTMENT. The partial key of the weak entity type SECTION is the attribute section name since, for each department, a section name uniquely identifies one single section for that department.

Though weak entity type is said to be existence dependent on the identifying entity type, however, not every existence dependency results in a weak entity type. For example, the PASSPORT entity type cannot exist unless it is related to a PERSON entity type, even though it has its own key attribute (passport_no) and hence is not a weak entity type.

A **double outlined rectangle** represents a weak entiry type. The partial key is represented by a **dotted underline**.

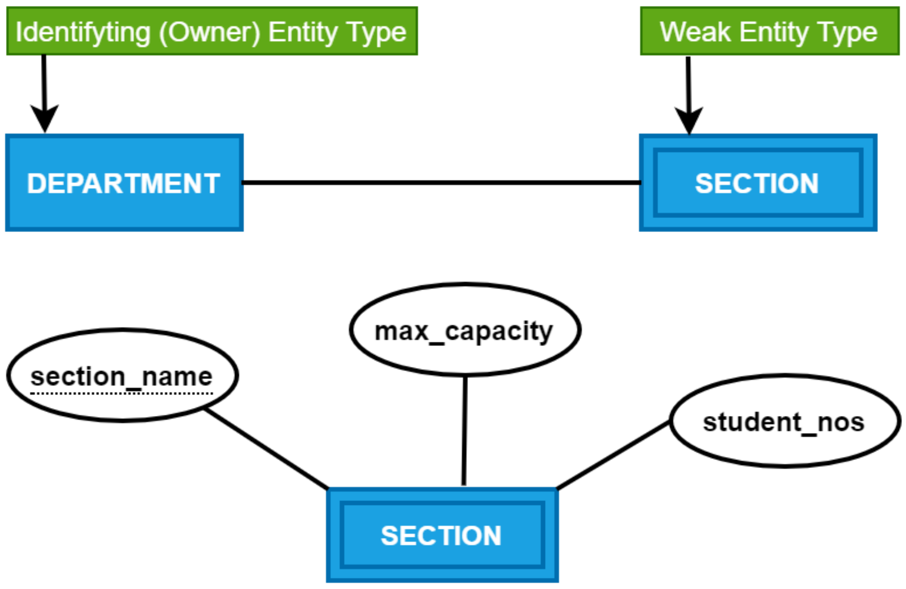


Figure 2.11 Representation of Weak Entity (SECTION)

What is relationship?

When two or more entity types are linked together to manage the related information, such linkage is called **relationship**, e.g., every STUDENT belongs to a DEPARTMENT. These two entities have a relationship.
Use a **diamond** symbol to create the relationship between the entity types.

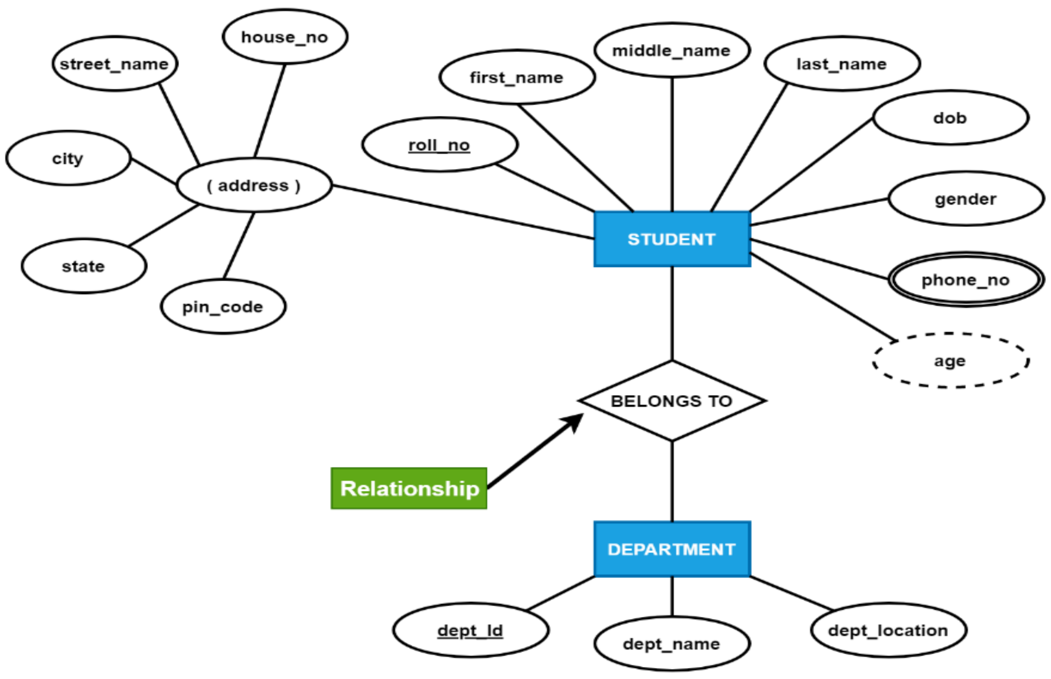


Figure 2.12 Representation of Relationship (BELONGS TO)

Relationship Types, Sets, and Instances

Relationship Type

When two or more entity types are linked together to manage the related information, then such linkage is called a **relationship type**, e.g., every STUDENT belongs to a DEPARTMENT. These two entities have a relationship type – "BELONGS TO"
Use the **diamond** symbol to create the relationship type between the entity types.

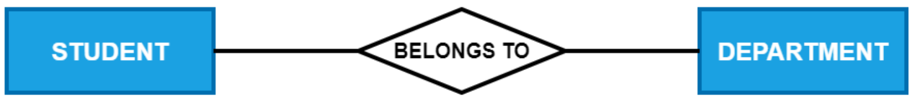
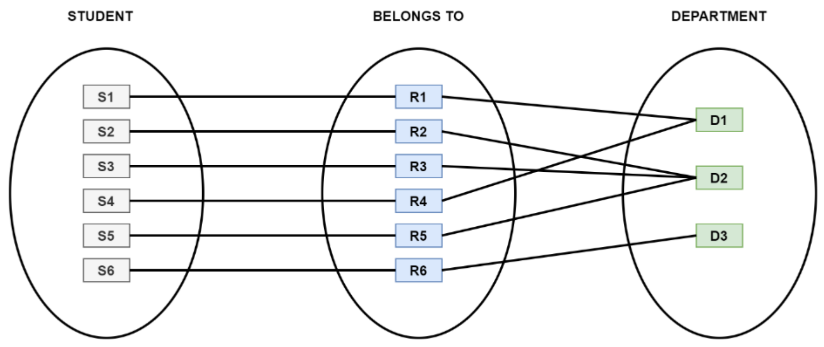
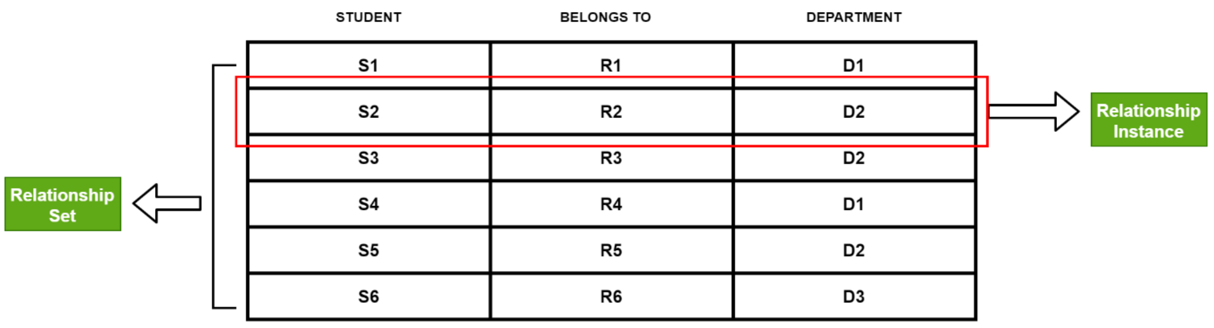


Figure 2.13 Representation of Relationship (BELONGS TO)

Relationship instance and a Relationship Set

A **relationship instance** is an instance that associates an entity instance from an entity type to another entity instance of another entity type to establish a relationship among various participating entity types.
A **relationship set** is the set of all relationship instances that participates in any relationship type to define a relationship between various participating entity types.




Relationship Set
Relationship Instance

STUDENT	BELONGS TO	DEPARTMENT
S1	R1	D1
S2	R2	D2
S3	R3	D2
S4	R4	D1
S5	R5	D2
S6	R6	D3

Figure 2.14 Representation of Relationship Set and Relationship Instance

Relationship Degree, Role Name, and Recursive Relationships

Degree of a Relationship Type

The **degree** of a relationship type is the number of participating entity types. Hence, the BELONGS TO relationship (in the section above) is of degree two has only two participating entity types, i.e., STUDENT and DEPARTMENT. A relationship of degree two is called **binary**. A relationship type with degree three is called a **ternary** relationship. An example of a ternary relationship type is COURSE ALLOCATION. The three participating entity types in COURSE ALLOCATION are DEPARTMENT, FACULTY, and COURSE. This relationship states the department-wise allocation of course and faculty.

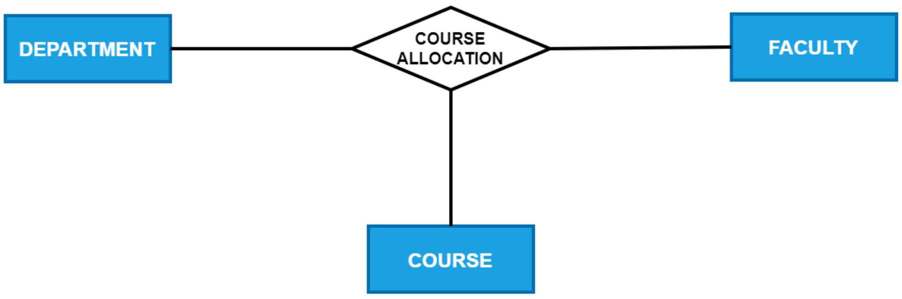
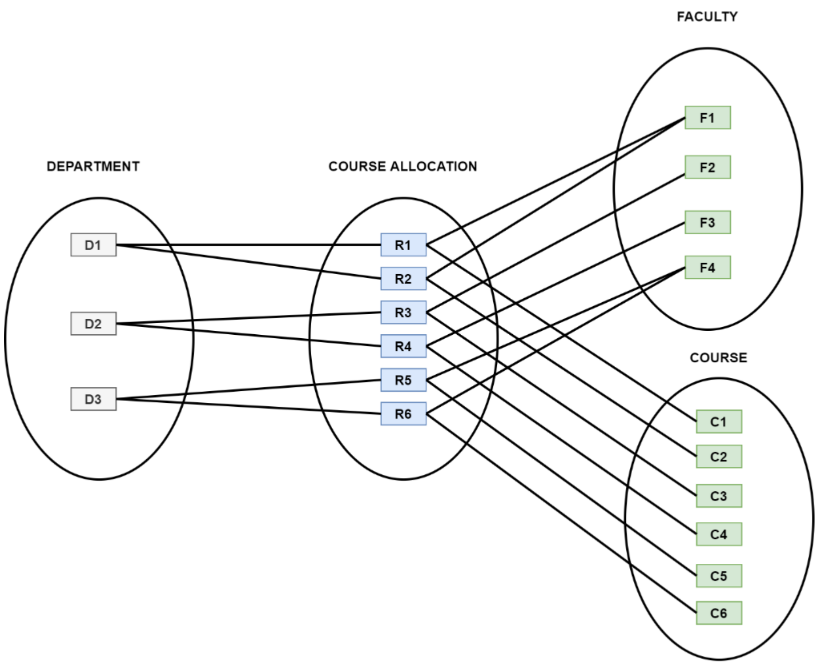


Figure 2.15 ER diagram with a ternary relationship (COURSE ALLOCATION)



DEPARTMENT	COURSE ALLOCATION	FACULTY	COURSE
D1	R1	F1	C1
D1	R2	F1	C2
D2	R3	F2	C3
D2	R4	F3	C4
D3	R5	F4	C5
D3	R6	F4	C6

Figure 2.16 Representation of a ternary relationship (COURSE ALLOCATION)

Role Names and Recursive Relationships

Each entity type that participates in a relationship type plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance and helps to explain what the relationship means.

For example, in the BELONGS TO relationship type, STUDENT plays the role of a student, and DEPARTMENT plays the role of an academic department.

Role names are not technically necessary in relationship types where all the participating entity types are distinct since each participating entity type name can be used as the role name. However, in some cases, the same entity type participates more than once in a relationship type in different roles. In such cases, the role name becomes essential for distinguishing the meaning of each participating entity's role. Such relationship types are called **recursive relationships**.

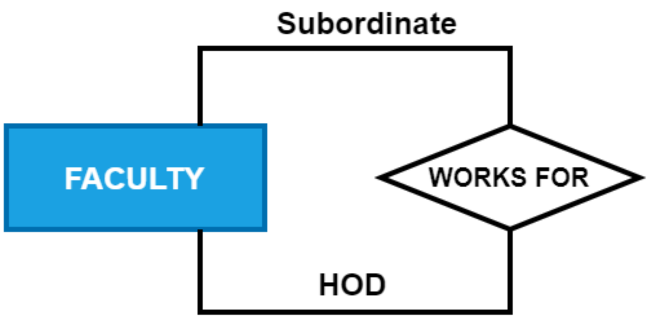
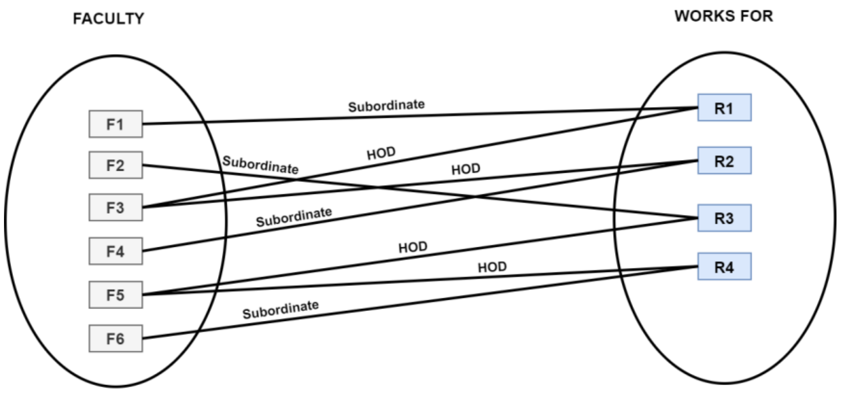


Figure 2.17 ER diagram with role indicators

The WORKS FOR **relationship type** relates a faculty to a HOD, where both faculty and HOD entities are members of the same FACULTY entity set. Every HOD is a faculty also. Hence, the FACULTY entity type participates twice in WORKS FOR relationship type, once in the role of a faculty and another in the role of a HOD.



SUBORDINATE	WORKS FOR	HOD
F1	R1	F3
F2	R3	F5
F4	R2	F3
F6	R4	F5

Figure 2.18 Representation of Recursive relationship (WORKS FOR)

Attribute attached to a relationship type

A relationship type can also have attributes attached with it which are called **descriptive attributes**.
Example:

COURSE ALLOCATION in

Figure 2.19 is done for a particular academic year and a semester. The figure below shows the ER diagram with **acad_yr** & **sem** as descriptive attributes associated with the relationship type COURSE ALLOCATION. Here acad_yr represents an academic year, and sem represents a semester. This relationship states the department-wise allocation of course and faculty in the particular academic year (acad_yr) and semester (sem).

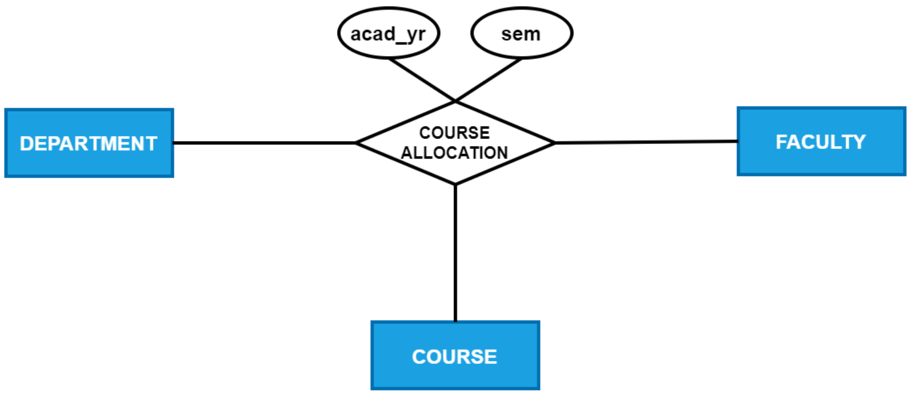


Figure 2.19 ER diagram of attribute attached with to a relationship type

2.3 Constraints on Binary Relationship Types

Relationship types usually have certain constraints that limit the possible combination of entities that may participate in the corresponding relationship set. These constraints are determined from the real-world situation that the relationships represent. For example, a student can be part of exactly one department, and then we would like to describe this constraint in the database design. We will now look at two main types of binary relationship constraints, **cardinality ratio** , and **participation**.

Cardinality Ratio

The cardinality ratio expresses the number of entities to which another entity can be associated via a relationship set.

For a binary relationship type R between entity types A and B, the mapping cardinality must be one of the following:

- **One-to-one**. An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

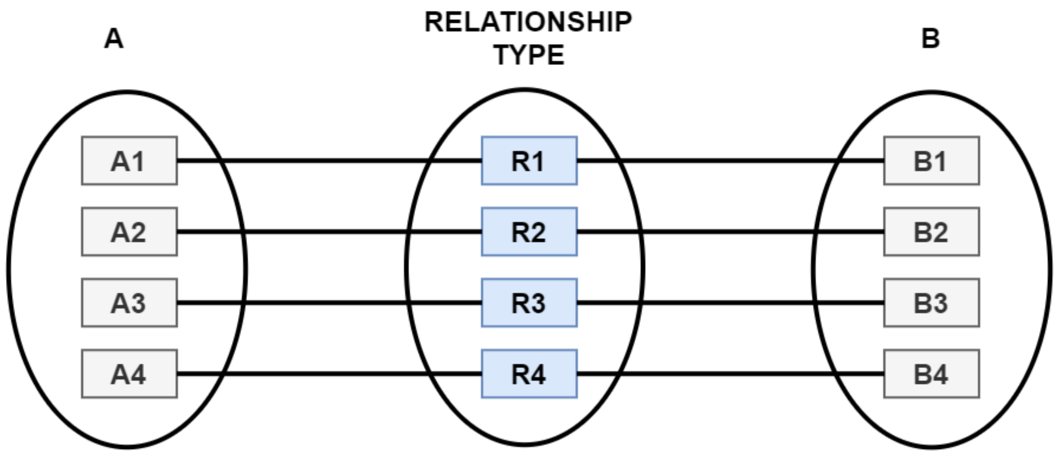


Figure 2.20 Representation of One-To-One Cardinality

Example:

Students do mini-projects as part of the curriculum. **"A student should opt only one mini-project, and one mini-project can be opted by one student only"**. Therefore there is one-to-one cardinality between two entity types, STUDENT and MINI PROJET, via OPTS relationship type.

A student must opt for only one of the mini-projects from the MINI PROJECT entity type. Hence the STUDENT entity set is associated with one cardinality limit to the MINI PROJECT entity set via OPS relationship set. To represent this, "1" is put on the opposite side of the STUDENT entity type (near MINI PROJECT entity type). Similarly, a mini-project will be assigned to only one student. Hence MINI PROJECT entity set is also associated with one cardinality limit to the STUDENT entity set, so "1" is put on the opposite side of MINI PROJECT (near to STUDENT entity type).

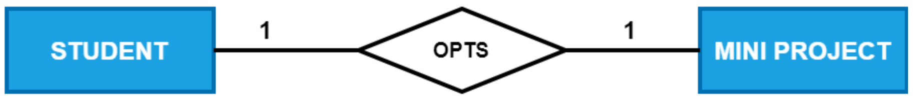


Figure 2.21 Entity-relationship showing One-To-One Cardinality

- **One-to-many**. An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.

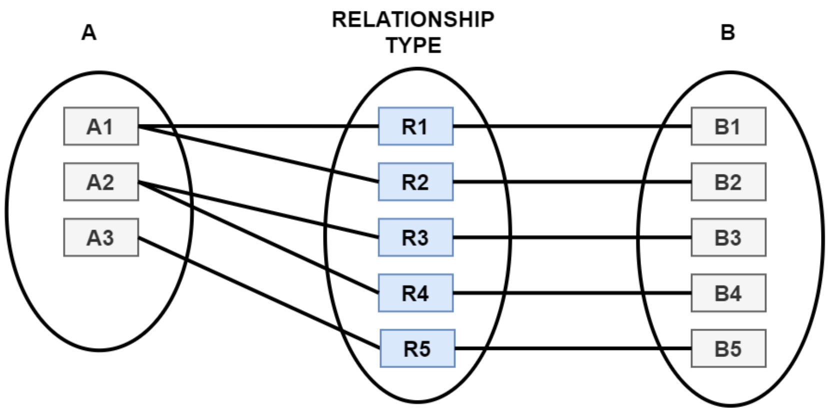


Figure 2.22 Representation of One-To-many Cardinality

Example:

“There are several departments, each department has many faculties, and one faculty can be a part of only one department”. There is one-to-many cardinality between the DEPARTMENT entity type and FACULTY entity type via HAS relationship type.

Each department has many faculty means that in one department, there can be several faculty members. Therefore DEPARTMENT entity set is associated with many cardinality limit to the FACULTY entity set via HAS relationship set. To represent this, "M" is put on the FACULTY side (opposite to the DEPARTMENT entity type). A faculty can report to one department only. Therefore, the FACULTY entity set is associated with one cardinality limit to the DEPARTMENT entity set, so “1” is put near the DEPARTMENT entity type (opposite to the FACULTY entity type).

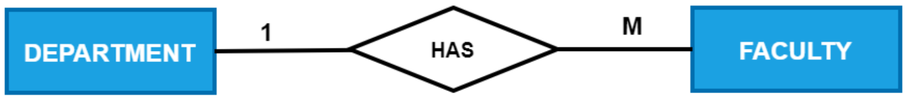


Figure 2.23 Entity relationship showing One-To-Many Cardinality

- **Many-to-one**. An entity in A is associated with at most one entity in B. However, an entity in B can be associated with any number (zero or more) of entities in A.

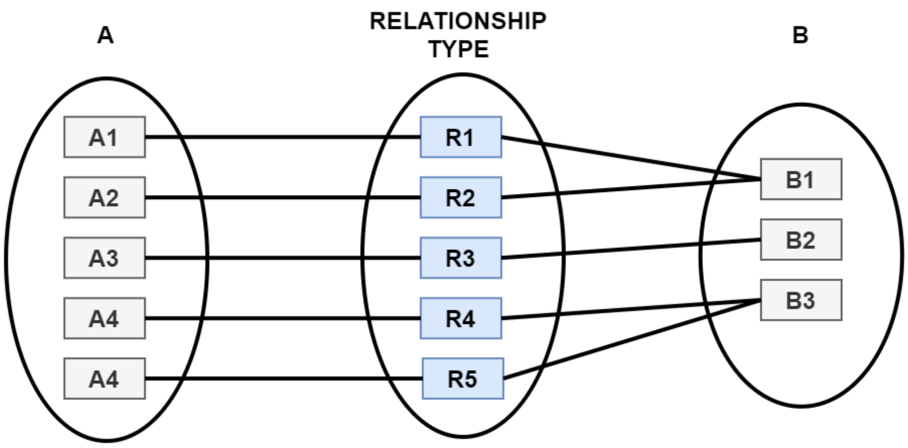


Figure 2.24 Representation of Many-To-One Cardinality

Example:

“Many students can belong to one department at a time”, there is a many-to-one cardinality ratio between STUDENT and DEPARTMENT entity type via BELONGS TO relationship type.

A student can belong to one department. Therefore, the STUDENT entity set is associated with one cardinality limit to the DEPARTMENT entity set via BELONGS TO relationship set. To represent this “1” is placed opposite to STUDENT entity type (near to DEPARTMENT entity type). Similarly, a department can have many students; hence DEPARTMENT entity set is associated with many cardinality limit to the STUDENT entity set, so "M" is placed opposite the DEPARTMENT entity type (near to STUDENT entity type).

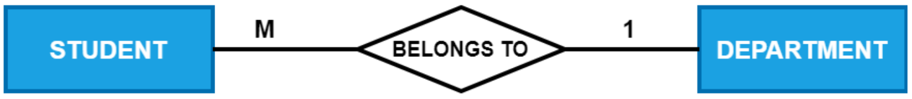


Figure 2.25 Entity relationship showing Many-To-One Cardinality

- **Many-to-many**. An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.
-

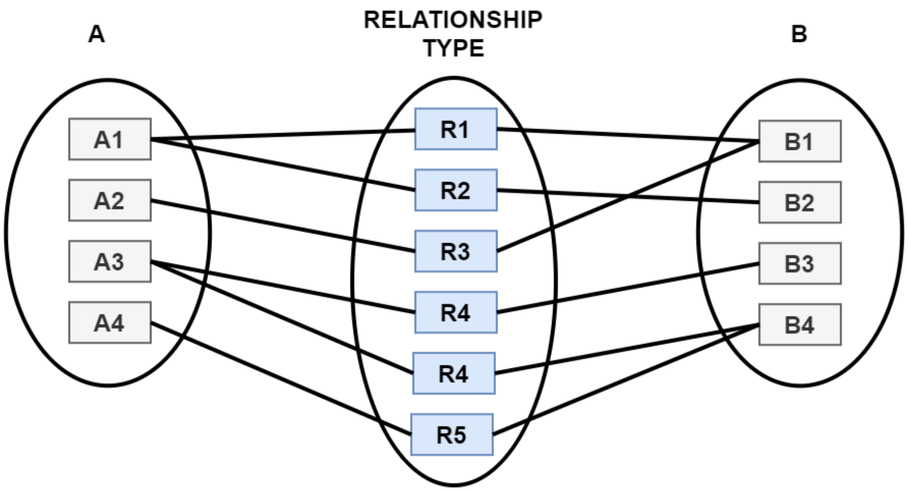


Figure 2.26 Representation of Many-To-Many Cardinality

Example:

"A faculty member can take multiple courses, and multiple faculty members can take one course", there is a many-to-many cardinality between FACULTY and COURSE entity types via TEACHES relationship type.

A faculty can teach multiple courses. Therefore, the FACULTY entity set is associated with many cardinality limit to the COURSE entity set via TEACHES relationship set. To represent this, "M" is placed opposite the FACULTY entity type (near COURSE entity type). Similarly, a course can be taught by multiple faculty members; hence COURSE entity set is associated with many cardinality limit to the FACULTY entity set, so "N" is placed opposite to the COURSE entity type (near to FACULTY entity type).

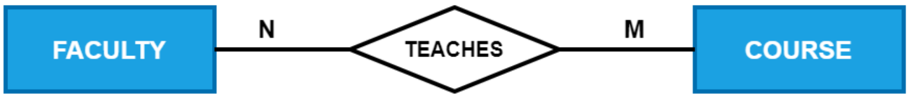


Figure 2.27 Entity-relationship showing Many -To- Many Cardinality

2.3.1 Limit on Cardinality Ratio

ER diagrams also provide a way to indicate more complex constraints on the number of times each entity participates in relationships in a relationship set. An edge between an entity set and a binary relationship set can have an associated minimum and maximum cardinality. This is shown in the form l..h, where l is the minimum and h the maximum cardinality.

Example:

Faculty are supposed to work on research projects. One faculty can work on Zero or a maximum of three research projects. A research project can have a minimum of one and a maximum of any number of faculty members.
The limit 0..3 on the line near the FACULTY entity type indicates that a faculty member can work on 0 or a maximum of 3 research projects at a time. Similarly, limit 1..* on the line near the RESEARCH PROJECT entity type indicates that a research project can have "1" or any number of faculty members working in that research project.

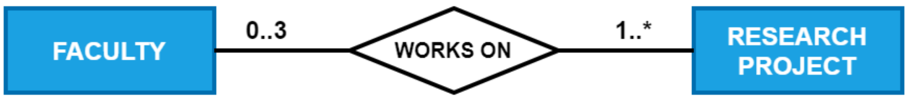


Figure 2.28 Entity-relationship showing Limits on Cardinality Ratio

2.3.2 Participation Constraints and Existence Dependencies

The participation of an entity set E in a relationship set R is said to be **total** if every entity instance in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be **partial**. Total participation is also called **existence dependency**.

In an ER diagram, total participation is shown by **double lines**, which indicates the total participation of an entity set in a relationship set & partial participation is shown by a **single line** which indicates the partial participation of an entity set in a relationship set.

Example:

Let's take the relationship between the DEPARTMENT entity type and the FACULTY entity type. A faculty has to be part of a department; therefore, the participation of FACULTY entity type in HAS relationship type is total and is shown by a double line. It indicates that for a faculty to exist, it must be associated with a department. A department has partial participation in HAS relationship type, as a department may not have even a single faculty when it is just established. So a department can exist without any faculty for some time and is shown by a single line in the ER diagram.

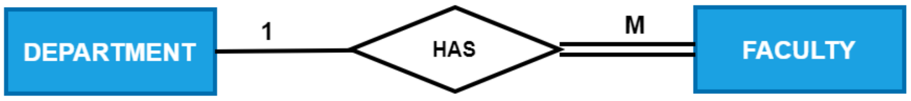


Figure 2.29 Entity relationship showing partial (single line) and total participation (double lines)

A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying entity type because a weak entiry type cannot exist without an owner (identifying) entity type. The relationship associating the weak entity type with its identifying entity type is called the **identifying relationship type**. Identifying relationship type is shown by a **double outlined diamond symbol**.

Example:

The SECTION entity type is a weak entity type as it does not have any key attributes. DEPARTMENT entity type helps to uniquely identify each entity instance of the SECTION entity type.

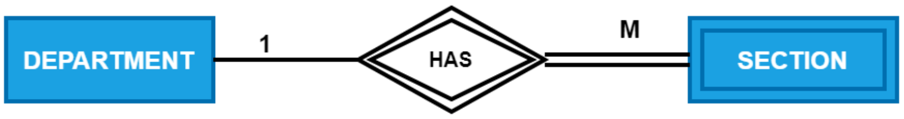


Figure 2.30 Entity relationship showing identifying relationship (double outlined diamond)

2.4 Keys

A key is an attribute or a set of attributes of an entity type that help to uniquely identify an entity instance in an entity set and is also used to establish relationships between the different entity types.

When we convert the ER model into a database structure, all entity types in the ER model get converted into **Tables (or Relations)**. The entiry type attributes are converted into **Table (or Relation) columns** and all the entity instances get converted into **Table (or Relation) rows**. These **Tables (or Relations)** have different types of keys. Two important types of keys are discussed here.

- Primary Key or Composite Primary Key
- Foreign Key

Primary Key or Composite Primary Key

A **Table column(s)** with the following set of features can be designated as the primary key. A **Table** can have one and only one primary key.

- Its value can never be NULL.
- Its value is always unique.
- The value once provided cannot be changed.

Example:

The roll_no of a student is a primary key that helps in identifying each student uniquely.
The dept_id of a department is also a primary key that helps in uniquely identifying a department.
If a **Table** does not have a single Table column that can identify Table rows uniquely, we have to combine two or more Table columns to make the key.
Combining two or more Table columns that can be used to identify Table rows uniquely is called the **composite key** or **composite primary key**.

Foreign Key

A primary key of a Table is called a foreign key to some other Table when that column is used to relate the two Tables.

Example:

Every student belongs to one department. To know the student department, we need to add dept_id to the STUDENT Table and create a relationship between DEPARTMENT and STUDENT Tables. In such a case DEPARTMENT Table will be called a **Referenced Table (Relation),** and the STUDENT Table will be called a **Referencing Table (Relation)**.
The Table column used as a foreign key in a Table (Relation) must be a primary key in another Table (Relation).
Detailed discussion on keys is covered in Chapter 3.

ER Diagram show relationship between DEPARTMENT and STUDENT

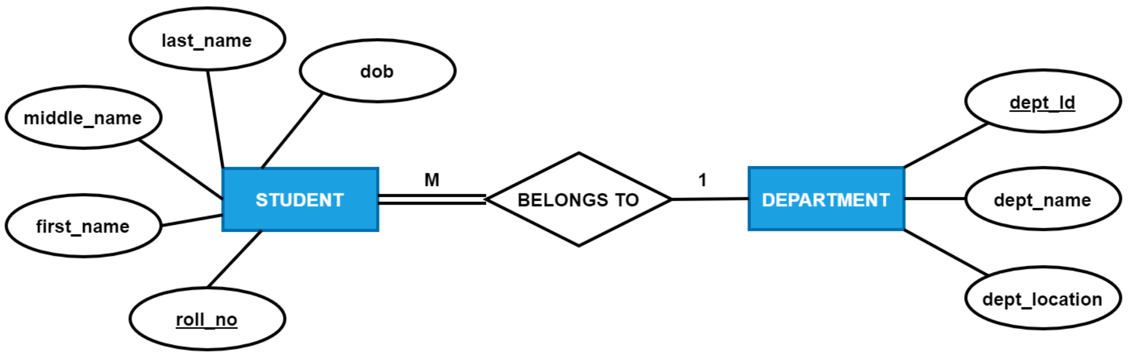
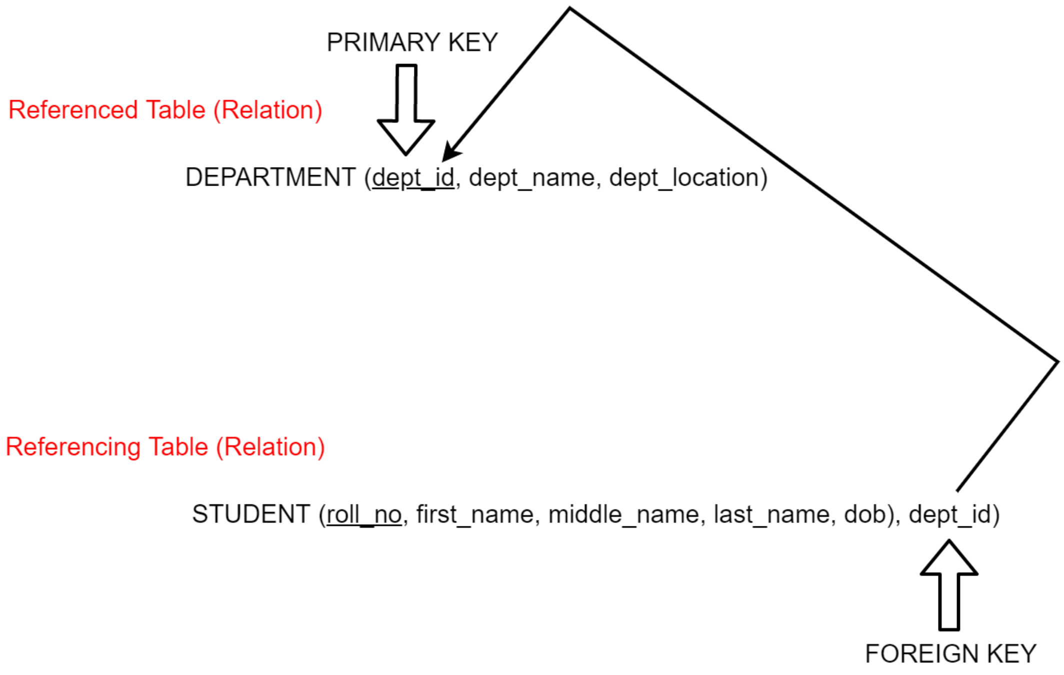


Figure 2.31 Entity-relationship showing relationship between DEPARTMENT and STUDENT

Converting the above ER diagram into Tables (Relations) and connecting them with primary and foreign key


PRIMARY KEY
Referenced Table (Relation)
DEPARTMENT (dept_id, dept_name, dept_location)
Referencing Table (Relation)
STUDENT (roll_no, first_name, middle_name, last_name, dob), dept_id)
FOREIGN KEY

Tabular view to connect primary key and foreign key

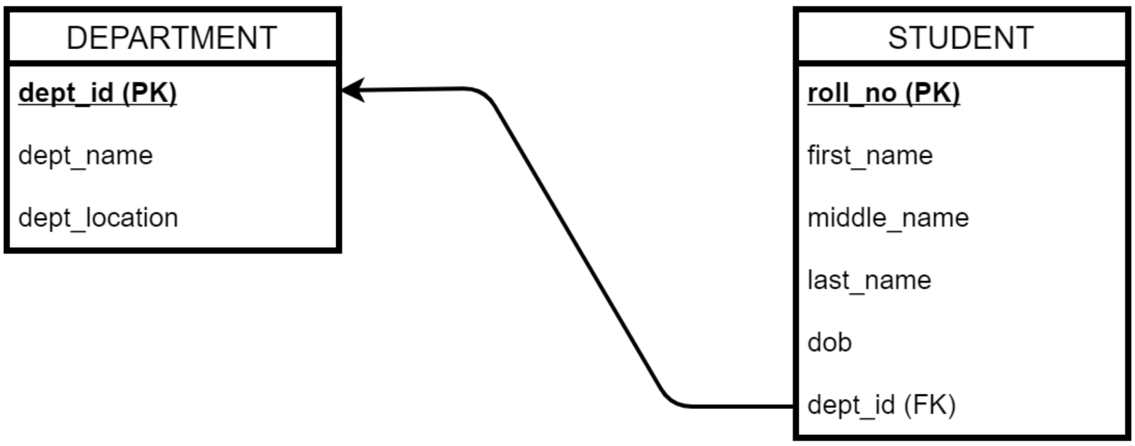


Figure 2.32 Representation to show the concept of Foreign Key (dept_id)

2.5 Extended E- R concepts

Although the basic ER concepts can model most database features, some aspects of a database may be more aptly expressed by certain extensions to the basic ER model. This section discusses the extended ER features of specialization, generalization, higher- and lower-level entity types, attribute inheritance, and aggregation.

2.5.1 Specialization

An entity type may include subgroupings of distinct entities in some way from other entities in the set. For instance, a subset of entities within an entity type may have attributes that are not shared by all entity types. The ER model provides a means for representing these distinctive entity groupings. Consider an entity type **EMPLOYEE** in the college, with attributes **employee id, first name, middle name, last name, gender, date of joining, designation, salary**. An employee may be further classified as one of the following:

- faculty
- staff

Each of these employee types is described by attributes that include all the attributes of entity type employee plus possibly additional attributes. For example, the **FACULTY** entity type may be described further by the attribute **Ph.D. status, the number of research papers published**. In contrast, the **STAFF** entity type may be described further by the attribute **technical or non-technical**. The process of designating subgroupings within an entity type is called specialization. The specialization of an employee allows us to distinguish among persons according to whether they are faculty or staff.

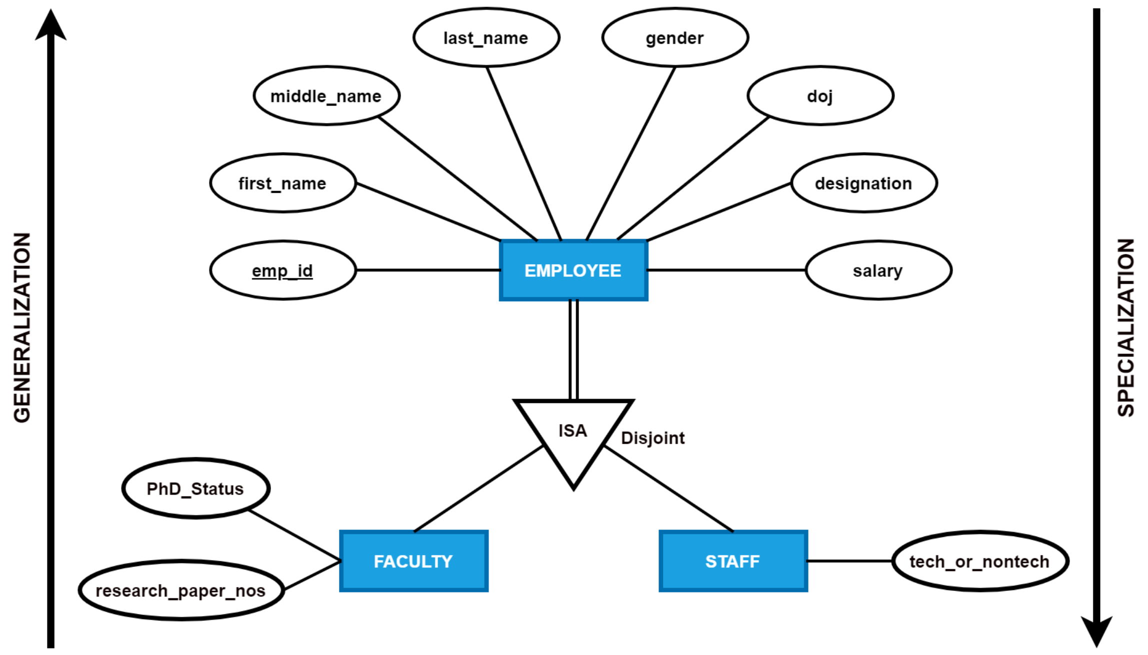


Figure 2.33 Specialization and Generalization of EMPLOYEE entity type

2.5.2 Generalization

The refinement from an initial entity type into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity types are synthesized into a higher-level entity type based on common features.
The database designer may have first identified a FACULTY entity type with the attributes **employee id, first name, middle name, last name, gender, date of joining, designation, salary, Ph.D. status, number of research papers published,** and a STAFF entity type with the attributes **employee id, first name, middle name, last name, gender, date of joining, designation, salary, technical or non-technical**. There are similarities between the **FACULTY** entity type and the **STAFF** entity type because they have several attributes in common.

This commonality can be expressed by generalization, a containment relationship between a higher-level entity type and one or more lower-level entity types.

In our example, the **EMPLOYEE** is the higher-level entity type and **FACULTY** and **STAFF** are lower-level entity types. Higher- and lower-level entity types also may be designated by the terms superclass and subclass, respectively. The **EMPLOYEE** entity type is the superclass of the **FACULTY** and **STAFF** subclasses. For all practical purposes, generalization is a simple inversion of specialization. We will apply both processes, in combination, in the course of designing the ER schema for an enterprise.

2.5.3 Attribute Inheritance

A crucial property of the higher- and lower-level entities created by specialization and generalization is attribute inheritance. The attributes of the higher-level entity types are said to be inherited by the lower-level entity types. For example, FACULTY and STAFF inherit the attributes of the EMPLOYEE. Thus, FACULTY is described by its **employee id, first name, middle name, last name, gender, date of joining, designation, salary** attributes, and additionally**, Ph.D. status, number of research papers published** attributes. STAFF is described by its **employee id, first name, middle name, last name, gender, date of joining, designation, salary** attributes, and additionally **technical or non-technical** attribute. A lower-level entity type (or subclass) also inherits participation in the relationship types in which its higher-level entity (or superclass) participates.
Whether a given portion of an ER model was arrived at by specialization or generalization, the outcome is the same:

- A higher-level entity type with attributes and relationships that apply to all of its lower-level entity types
- Lower-level entity types with distinctive features that apply only within a particular lower-level entity type

2.5.4 Constraints on generalization

There are certain constraints that database designers choose to put on a particular generalization.

One type of constraint involves determining which entities can be members of a given lower-level entity type. Such membership may be one of the following:

- **Condition-defined**: In condition-defined constraints lower-level entity types, membership is evaluated based on whether or not an entity satisfies an explicit condition or predicate. For example, assume that the higher-level entity type **EMPLOYEE** has the attribute **employee-type**. All employee entities are evaluated on the defining employee-type attribute. Only those entities that satisfy the condition employee-type = "FACULTY" are allowed to belong to the lower-level entity type FACULTY. All entities that satisfy the condition employee-type = "STAFF" are included in STAFF. Since all the lower-level entities are evaluated based on the same attribute (in this case, on account-type), this type of generalization is said to be attribute-defined.
- **User-defined**: User-defined lower-level entity types are not constrained by a membership condition; rather, the database user assigns entities to a given entity type. For instance, let us assume that employees are assigned to one of four work teams after three months of employment. We, therefore, represent the teams as four lower-level entity types of the higher-level EMPLOYEE entity type. A given employee is not assigned to a specific team entity automatically based on an explicit defining condition. Instead, the user in charge of this decision makes the team assignment on an individual basis. The assignment is implemented by an operation that adds an entity to an entity type.

The second type of constraint relates to whether or not entities may belong to more than one lower-level entity type within a single generalization. The lower-level entity types may be one of the following:

- **Disjoint**: A disjointness constraint requires that an entity belongs to no more than one lower-level entity type. In our example, an EMPLOYEE entity can satisfy only one condition; an entity can be either a faculty or a staff, but cannot be both.
- **Overlapping**: In overlapping generalizations, the same entity may belong to more than one lower-level entity type within a single generalization. For an illustration, consider the employee work team example, and assume that certain managers participate in more than one work team. Therefore, a given employee may appear in more than one of the team entity types that are lower-level entity types of the EMPLOYEE. Thus, the generalization is overlapping.

Lower-level entity overlap is the default case; a disjointness constraint must be placed explicitly on a generalization (or specialization). We can note a disjointedness constraint in an ER diagram by adding the word disjoint next to the triangle symbol.

The final constraint, the completeness constraint on a generalization or specialization, specifies whether or not an entity in the higher-level entity type must belong to at least one of the lower-level entity types within the generalization/specialization. This constraint may be one of the following:

- Total generalization or specialization. Each higher-level entity must belong to a lower-level entity type.
- Partial generalization or specialization. Some higher-level entities may not belong to any lower-level entity type.

Partial generalization is the default. We can specify total generalization in an ER diagram using a double line to connect the box representing the higher-level entity type to the triangle symbol. (This notation is similar to the notation for total participation in a relationship.)
The employee generalization is total: All employee entities must be either be a faculty or staff. Because the higher-level entity type arrived at through generalization is generally composed of only those entities in the lower-level entity types, the completeness constraint for a generalized higher-level entity type is usually total. When the generalization is partial, a higher-level entity is not constrained to appear in a lower-level entity type. The work team entity types illustrate a partial specialization. Since employees are assigned to a team only after three months on the job, some employee entities may not be members of any lower-level team entity types.

2.5.5 Aggregation

One limitation of ER modeling is that it cannot express the relationships among relationships. To illustrate the need for this, aggregation is used. Aggregation is an abstraction through which relationships with its corresponding entities are aggregated into a higher-level entity.

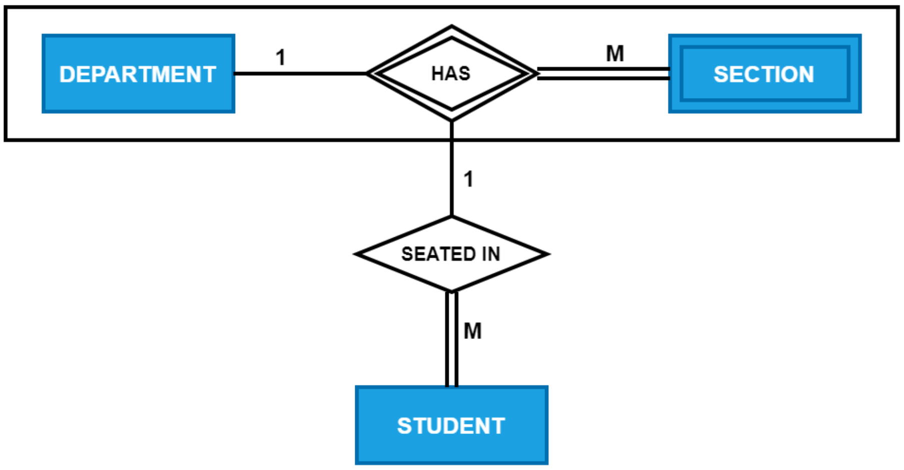


Figure 2.34 ER Diagram to show aggregation

As shown in above Figure 2.34, A department has many sections and a student needs to be seated in a section of the enrolled department. So, SEATED IN relationship is needed between the relationship HAS and entity type STUDENT. Using aggregation, HAS relationship with its entity types DEPARTMENT and SECTION is aggregated into a single entity and the relationship SEATED IN is created between the aggregated entity and STUDENT entity type.

2.6 Completion of ER Diagram – College Case Study

As we now understand the requirements from the user (pls. refer 2.1), we will develop the ER diagram of the requirements we have understood.
Below are the four steps in designing an ERD for a DBMS.

1. Identify Entity
2. Decide relationships and cardinality
3. Draw Entities and relationships separately
4. Connect relationships and entities

2.6.1 Identify Entity

Identifying **nouns** in the below statement and make them bold Italic characters.
In a College, there are several **departments,** and each department has one head of the department (HOD). Department has a name, its location, and **students** that belong to the department. A student can belong to only one department, and a department can have many students. If a department has recently come into existence, it might not have any students. Students have roll number, name, date of birth, gender, hobby, phone number, and address. **Faculty** has a name, designation, date of joining, gender & salary and belongs to a particular department. A department can run many **courses** with assigned credits, and students can study any number of courses being offered by various departments. *There are **sections** within each department, and each section has many students. Each section has its name, maximum capacity of that section, and the number of students in that section. Students need to do one **mini-project** individually. Each available mini-project has to be chosen based on its name, domain, subject, and description.* Faculty members can teach multiple courses in multiple departments. One course can be taught by many faculty members across departments. Faculty members can be part of multiple **research projects**. These projects are either sponsored by the government, industry, or the college itself. The faculty member can do one or more projects, and one project can have more than one faculty member. Research projects have fixed duration, and their status needs to be tracked regularly.

2.6.2 Decide Relationships and Cardinalities

Identify **verbs** and make them bold Italic characters.
In a college, there are several departments, and each department **has** one head of the department (HOD). Department has a name, its location, and students that **belong** to the department. A student can **belong** to only one department, and a department can **have** many students. If a department has recently come into existence, it might not **have** any students. Students **have** roll number, name, date of birth, gender, hobby, phone number, and address. Faculty has a name, designation, date of joining, gender & salary and belongs to a particular

department. A department can **run** many courses with assigned credits and students can **study** any number of courses being offered by various departments. *There are* sections *within each department, and each section **has** many students. Each section has its name, max capacity* of that section, and the number of students in that section. Students need to **do** one mini-project individually*. Each available mini-project has to be chosen based on its name, domain, subject, and description.* Faculty members can **teach** multiple courses in multiple departments. One course can be **taught** by many faculty members across departments. Faculty members can **be part** of multiple research projects. These projects are either **sponsored** by the government, industry, or the college itself. The faculty member can do one or more projects, and one project can **have** more than one faculty member. Research projects have fixed duration, and their status needs to be tracked regularly.

2.6.3 Entities & Attributes

A **student** has a roll number, first name, middle name, last name, date of birth, hobby, gender, phone numbers, and address.

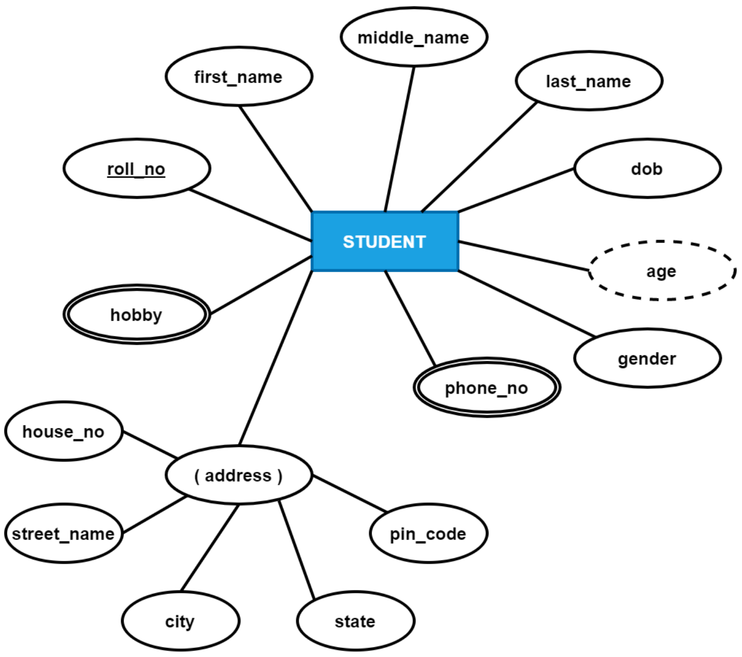


Figure 2.35 STUDENT entity type and its attributes

Faculty has faculty id, first name, middle name, last name, gender, date of joining, designation, and salary.

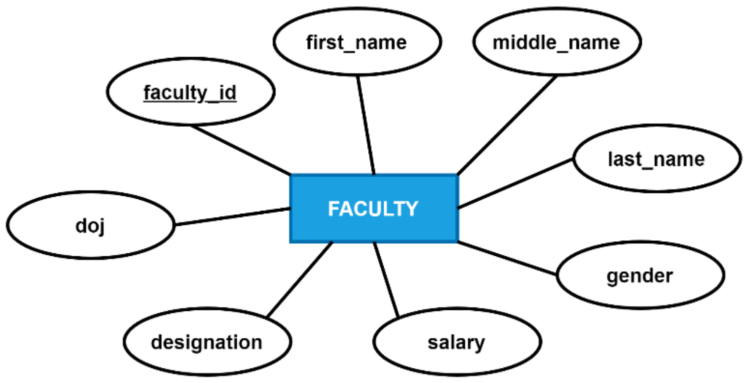


Figure 2.36 FACULTY entity type and its attributes

Course has course id, course name & course credit.

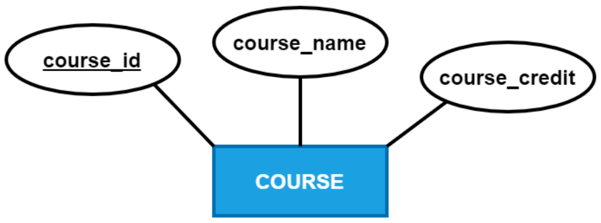


Figure 2.37 COURSE entity type and its attributes

Department has department id, department name, and department location.

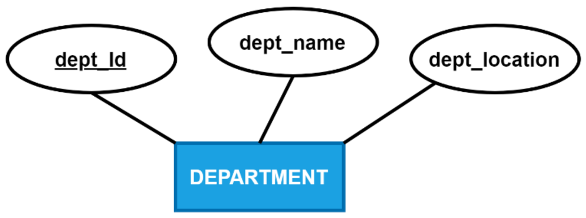


Figure 2.38 DEPARTMENT entity type and its attributes

Research Project has research project id, research project name, sponsoring agency duration, and status.

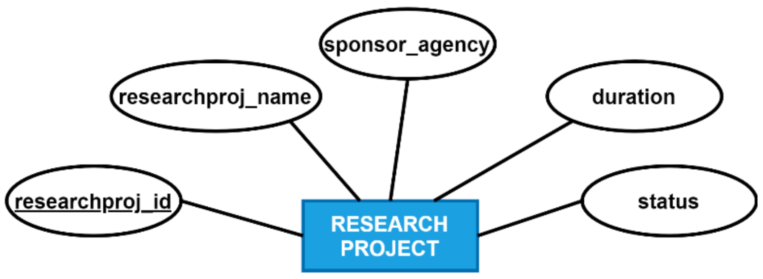


Figure 2. 39 RESEARCH PROJECT entity type and its attributes

Section has name, max capacity and the number of students in that section.

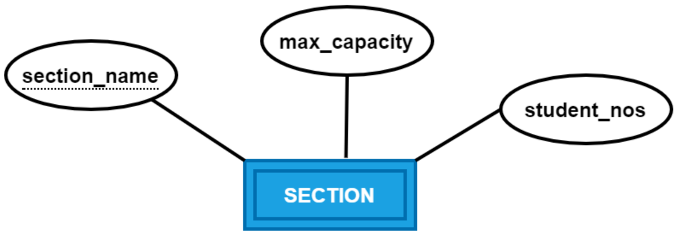


Figure 2.40 SECTION entity type and its attributes

Mini project has project id, project name, domain subject and description.

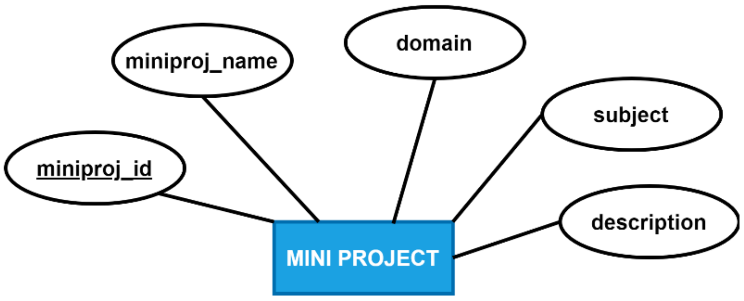


Figure 2.41 MINI PROJECT entity type and its attributes

2.6.4. Connect relationship and entities

Statement – A faculty can teach zero or more courses and course can be taught by zero or more faculty.

Relationship- Teaches

Cardinality- M : N

Participation- Partial for FACULTY entiry type (a new faculty might not have been allocated a course yet) and Partial for COURSE entiry type (a new course has not been opted by any faculty yet).

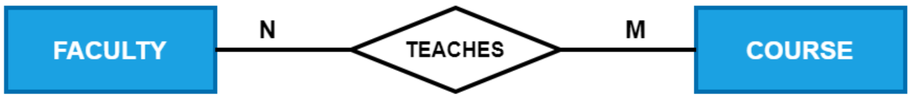


Figure 2.42 Representation of partial relationship between FACULTY and COURSE

Statement –A student belongs to one department, and a department can have zero or more students

Relationship- Belongs To

Cardinality- M : 1

Participation- Total for STUDENT entiry type (a student has to belong to a department to exist) and Partial for DEPARTMENT entiry type (a new department might not have any student yet).

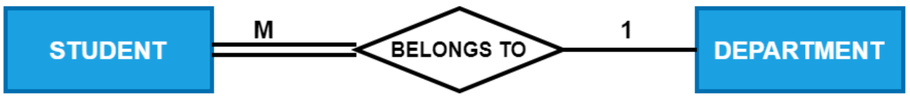


Figure 2.43 Representation of Strong and Partial relationship between STUDENT and DEPARTMENT

Statement – A department can have zero or more sections; a section must belong to a department. SECTION is a weak entity type as it does not have any prime attribute. The same section name can exist in all departments. Therefore the relationship between department and section is an identifying relationship. A section is identified by it's department.
Relationship- Has (Identifying Relationship)
Cardinality- 1 : M
Participation- Partial for DEPARTMENT entiry type (a new department might not have sections created yet) and Total for SECTION entiry type (section has no existence without department).

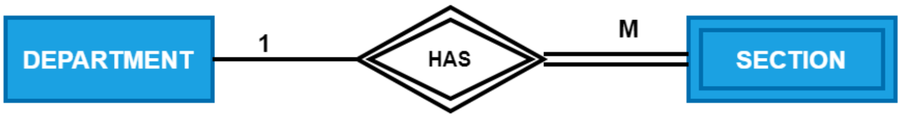


Figure 2.44 Representation of Partial and Total relationship between SECTION and DEPARTMENT

Statement –A department can have zero or more faculty. Every faculty must be part of only one department.
Relationship- Has
Cardinality- 1 : M
Participation- Partial for DEPARTMENT entiry type (a new department might not have any faculty yet) and Total for FACULTY entiry type (a faculty has to belong to a department to exist).

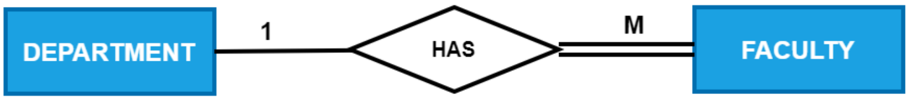


Figure 2.45 Representation of Total and Partial relationship between FACULTY and DEPARTMENT

Statement – Each student has to do one mini project individually and every available mini-project has to be opted by a student.
Relationship- Opts
Cardinality- 1 : 1
Participation- Total for STUDENT entiry type (a student has to opt a mini-project) and Total for MINI PROJECT entiry type (a mini project has to be opted by a student).

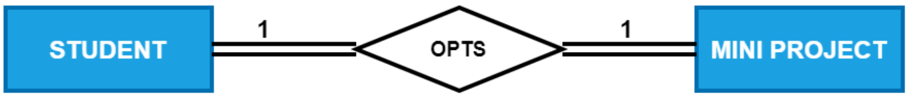


Figure 2.46 Representation of Total relationship between STUDENT and MINI PROJECT

Statement – A department can run zero or more courses, and each course must belong to only one department.

Relationship- Runs

Cardinality- 1 : M

Participation- Partial for DEPARTMENT entiry type (a new department might not be running any course yet) and Total for COURSE entiry type (a course has to belong to a department to exist).

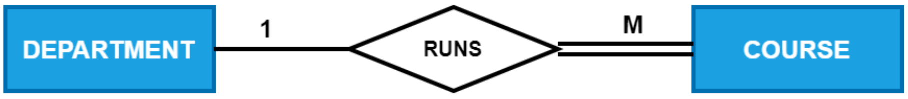


Figure 2.47 Representation of Total and Partial relationship between COURSE and DEPARTMENT

Statement – A section can have zero or more students, and each student must belong to any one section.

Relationship- Seated In

Cardinality- M : 1

Participation- Total for STUDENT entiry type (student has to be seated in a section) and Partial for SECTION entiry type (a new section might not have been allocated a student yet).

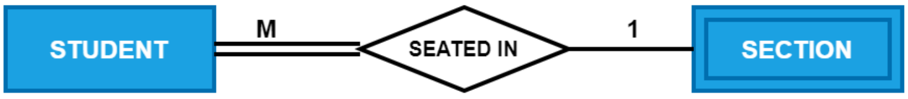


Figure 2.48 Representation of Total and Partial relationship between STUDENT and SECTION

Statement – Each student studies one or more courses as part of the curriculum, and each course can be opted by zero or more students.

Relationship- Studies

Cardinality- M : N

Participation- Total for STUDENT entiry type (a student has to study a course) and Partial for COURSE entiry type (a new course might not have been opted by a student yet).

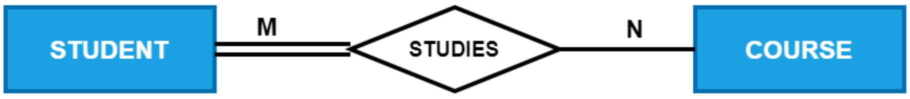


Figure 2.49 Representation of Total and Partial relationship between STUDENT and COURSE

Statement – A faculty may work on many research projects, and each research must have one or more faculty are working on it.

Relationship- Works On

Cardinality- M : N

Participation- Partial for FACULTY entiry type (a faculty may work on a research project) and Total for RESEARCH PROJECT entiry type (each research project needs to have faculty working on it).

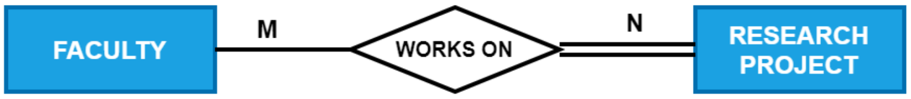


Figure 2.50 Representation of Total and Partial relationship between FACULTY and RESEARCH PROJECT

Statement –Faculty will have a HOD who is a faculty himself/herself. A HOD can have one or more faculty under him/her.

Relationship- Works For

Cardinality- M : 1

Participation- Total for HOD Role (HOD will have faculty under him/her) and Partial for Subordinate Role (The faculty who is a HOD will not have a HOD for him/her).

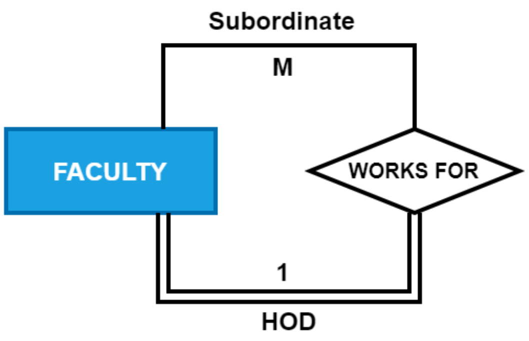


Figure 2.51 Representation of Total and Partial in FACULTY relationship as different Role

Complete ER Diagram-

Based on our understanding of entities, attributes, relationships & cardinalities, and ER Notations, below is the complete ER diagram of our case study.

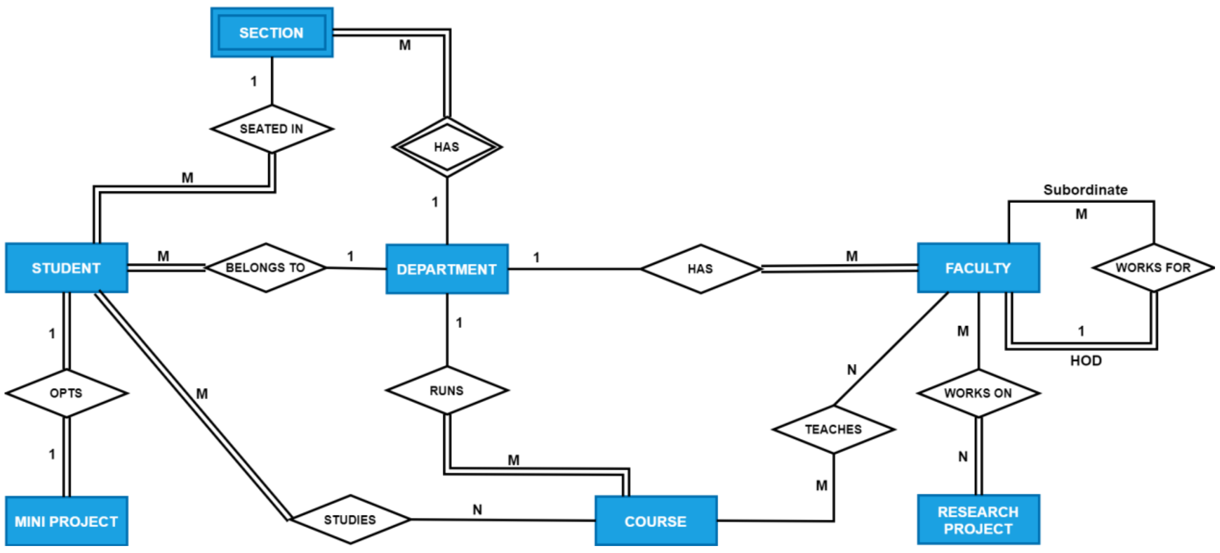


Figure 2.51a Complete ER diagram of Case Study

2.7 Reduction of an ER Model to Relational Tables

2.7.1 Introduction of Relational Database

A **Relational database** is the type of database based on the relational model proposed by E.F. Codd in 1970. A relational database is a collection of Relations. E.F. Codd has originally described a **Relation** as a set of tuples/records (t_1, t_2,…, t_n) where each element t_i belongs to a domain D_i. The domain is a set of values that an attribute can hold. For example, the domain for attribute roll_no of an entity type STUDENT is the set of all roll numbers. A relational database of any organization or what we can call a mini-world stores information about all its entity types in Relations. These Relations are also called **Tables** and each Relation is assigned a unique name in the database. A Relation or Table is shown in Figure 2.52, in which a row keeps the record of an entity instance and each column holds the values of an attribute for the complete entity set. A row in a Relation represents the relationship among values of different attributes for an entity instance. A software used to manage the relational database is called a **Relational Database Management System (RDBMS).**

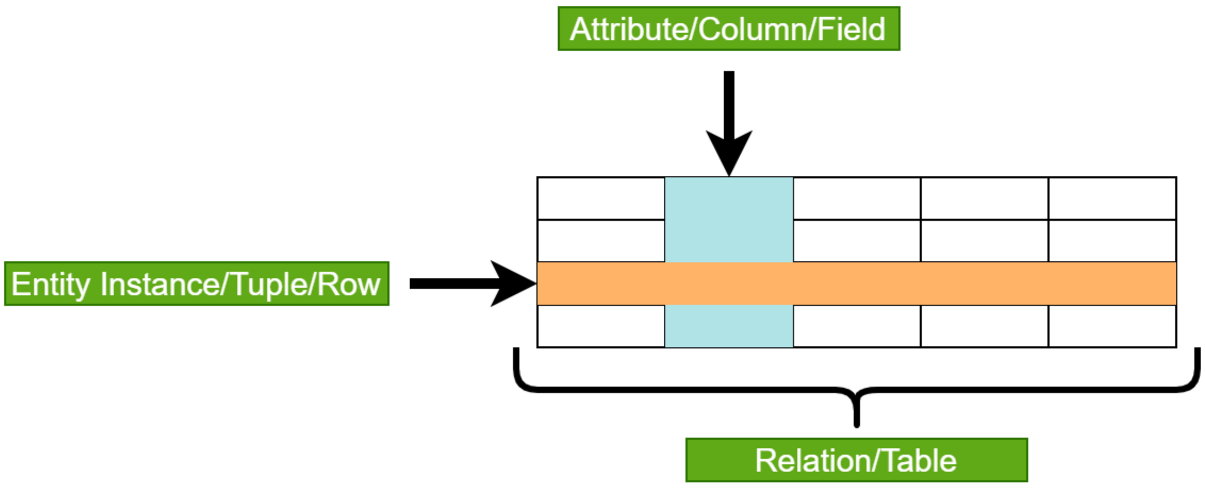
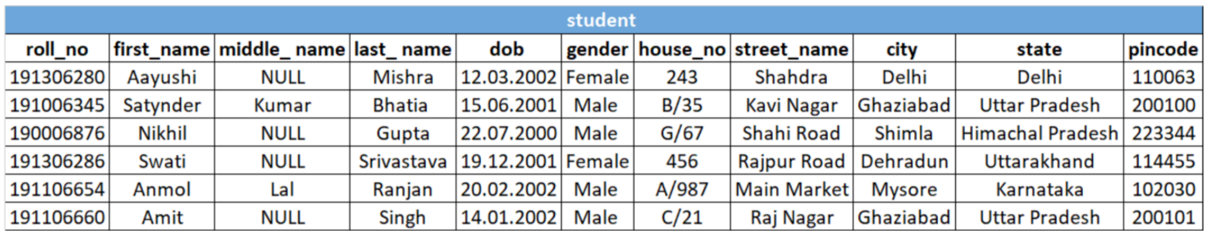


Figure 2.52 Representation of a Relation

A Table as shown in Figure 2.53 shows a Relation named STUDENT. It keeps a record of five students in five rows, which are also called tuples. This Relation has eleven attributes named roll_no, first_name, middle_name, last_name, dob, gender, house_no, street_name, city, state, pin-code.



student										
roll_no	first_name	middle_ name	last_ name	dob	gender	house_no	street_name	city	state	pincode
191306280	Aayushi	NULL	Mishra	12.03.2002	Female	243	Shahdra	Delhi	Delhi	110063
191006345	Satynder	Kumar	Bhatia	15.06.2001	Male	B/35	Kavi Nagar	Ghaziabad	Uttar Pradesh	200100
190006876	Nikhil	NULL	Gupta	22.07.2000	Male	G/67	Shahi Road	Shimla	Himachal Pradesh	223344
191306286	Swati	NULL	Srivastava	19.12.2001	Female	456	Rajpur Road	Dehradun	Uttarakhand	114455
191106654	Anmol	Lal	Ranjan	20.02.2002	Male	A/987	Main Market	Mysore	Karnataka	102030
191106660	Amit	NULL	Singh	14.01.2002	Male	C/21	Raj Nagar	Ghaziabad	Uttar Pradesh	200101

Figure 2.53 The STUDENT Relation (6 entity instances and 11 attributes)

A Relation has mainly two components:

1. Relation schema
2. Relation instance

1. Relation schema

A Relation schema describes the Relation (Table) structure as it includes the Relation name, names of attributes.

For example, the Relation schema of two Relations FACULTY and COURSE can be written as:

- FACULTY (faculty_id, first_name, middle_name, last_name, designation, doj, gender, salary)
- COURSE (course_id, course_name, course_credit)

The degree or arity of a Relation is the number of attributes in a Relation schema. For example, the degree of Relation schema FACULTY is eight, and the degree of Relation schema COURSE is three.

2. Relation instance

The snapshot of data stored in a Relation at any instant of time is called a Relation For example, an instance of the Relation STUDENT is shown in

Figure 2.

faculty							
faculty_id	first_name	middle_ name	last_ name	designation	doj	gender	salary
1231	Ram	Mohan	Prasad	Assistant Professor	17.12.2010	Male	54000
9765	Laxman	NULL	Naryan	Associate Professor	19.09.2013	Male	72000
1987	Ayush	NULL	Giri	Professor	13.01.2012	Male	100125
2765	Grish	Kumar	Sharma	Assistant Professor	01.02.2013	Male	62000
1734	Nidhi	NULL	Bhatia	Assistant Professor	06.07.2011	Female	56000

course		
course_id	course_name	course_credit
KCS301	Data Structure	3
KCS401	DBMS	3
KME502	Machine Design	2
KEC301	Digital Electronics	2
KAS101	Physics	3

department		
dept_id	dept_name	dept_location
D01	CS	Aryabhatta Block
D02	CSE	Aryabhatta Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D05	ME	Ramanujan Block

student											
roll_no	first_name	middle_ name	last_ name	dob	gender	house_no	street_name	city	state	pincode
191306280	Aayushi	NULL	Mishra	12.03.2002	Female	243	Shahdra	Delhi	Delhi	110063
191006345	Satynder	Kumar	Bhatia	15.06.2001	Male	B/35	Kavi Nagar	Ghaziabad	Uttar Pradesh	200100
190006876	Nikhil	NULL	Gupta	22.07.2000	Male	G/67	Shahi Road	Shimla	Himachal Pradesh	223344
191306286	Swati	NULL	Srivastava	19.12.2001	Female	456	Rajpur Road	Dehradun	Uttarakhand	114455
191106654	Anmol	Lal	Ranjan	20.02.2002	Male	A/987	Main Market	Mysore	Karnataka	102030
191106660	Amit	NULL	Singh	14.01.2002	Male	C/21	Raj Nagar	Ghaziabad	Uttar Pradesh	200101

Figure 2.54 A Relational Database that stores information of FACULTY, STUDENT, COURSE and DEPARTMENT (An instance of relational database: one possible state).

A relational database is shown in
Figure 2.. It has four Relations STUDENT, FACULTY, COURSE, and DEPARTMENT. The data shown here depicts the current state of the above Relations in the database. When the data is initially populated in the database, it is called the initial state. Going forward, tuples or records can be added, deleted, or updated in any Relation of the database. Any kind of modification will change the state of the Relation in the database. The state of Relations in a database can change frequently, but the schema of the Relations in the database is not supposed to change frequently.

The collection of data stored in the database at any instant of time is called a **database instance.** The overall design or structure of the database is called the **database** relational database schema is the collection of relation schemas and is shown in
Figure 2.**42**

FACULTY (faculty_id, first_name, middle_ name, last_ name, designation, doj, gender, salary)

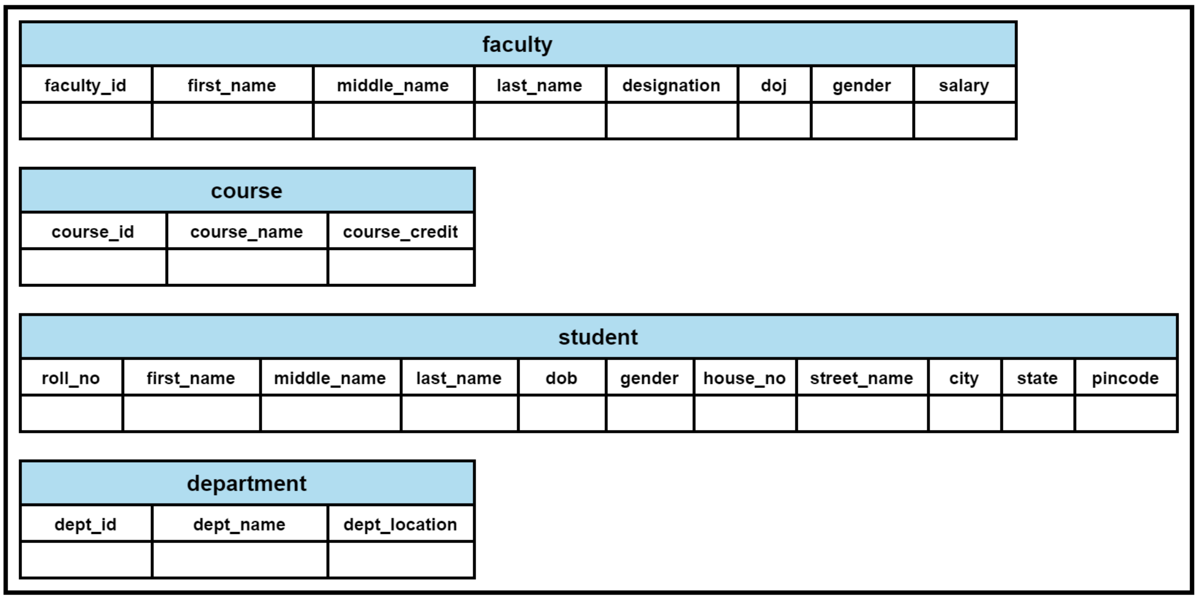
COURSE (course_id, course_name, course_credit)

STUDENT (roll_no, first_name, middle_ name, last_ name, dob, gender, house_no, street_name, city, state, pincode)

DEPARTMENT (dept_id, dept_name, dept_location)

Figure 2.42 Database Schema

or it can also be represented as:



faculty							
faculty_id	first_name	middle_name	last_name	designation	doj	gender	salary

course		
course_id	course_name	course_credit

student										
roll_no	first_name	middle_name	last_name	dob	gender	house_no	street_name	city	state	pincode

department		
dept_id	dept_name	dept_location

Figure 2.43 A Relational Database Schema diagram

After designing the conceptual model of the database using an ER diagram, we need to convert the conceptual model into the relational model, which can be implemented using any RDBMS, like Oracle SQL, MySQL etc.
Let's see what a Relational Model is.

Relational Model

The relational model represents how data is stored in Relational Databases. A relational database stores data in the form of Relations (Tables). In the relational model, entity types and their corresponding relationships are represented by a collection of inter-related Tables. Each Table is a group of columns and rows, where a column represents an attribute of an entity type and rows represent its instance.
Following rules are used for converting an ER diagram into the relational tables:

2.7.2 Tabular Representation of Regular (Strong) Entity types

For each regular (strong) entity type E in the ER schema, create a Relation R that includes all the simple attributes of E.

- A regular (strong) entity type with only simple attributes will be converted into a single Relation (Table).
- Attributes of the strong entity type will become the fields (column) of Relation with their respective data types.
- The key attribute of the entity type will become the primary key of the Table.

For Example:-

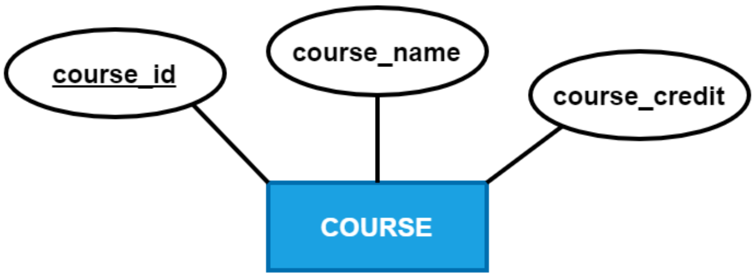


Figure 2.44 Representation of Strong Entity with simple attributes

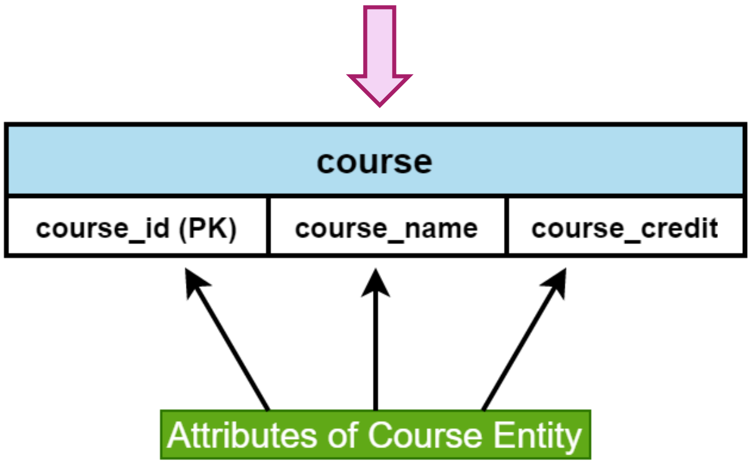


Figure 2.45 Representation of Strong Entity with simple attributes

Schema: COURSE (<u>course_id (PK)</u> , course_name, course_credit)

2.7.3 Tabular Representation of Weak Entity types

For each weak entity type W in the ER schema with owner entity type E, create a Relation R, and include all simple attributes (or simple components of composite attributes) of W as attributes of R. In addition, have as foreign key attributes of R, the primary key attribute(s) of the Relation(s) that correspond to the owner entity type(s). This takes care of the identifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W.

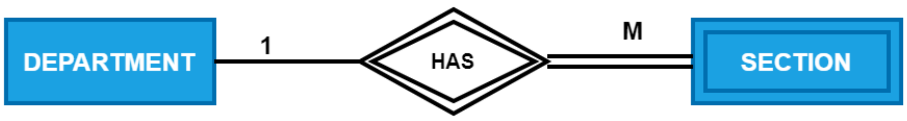


Figure 2.59 Weak entity type SECTION and its owner entity type DEPARTMENT

As shown in ER diagram in Figure 2.59, SECTION is a weak entity type and its owner entity type is DEPARTMENT; the Relation schema of SECTION will be represented as follows:

SECTION (section_name, max_capacity, student_nos, dept_id)

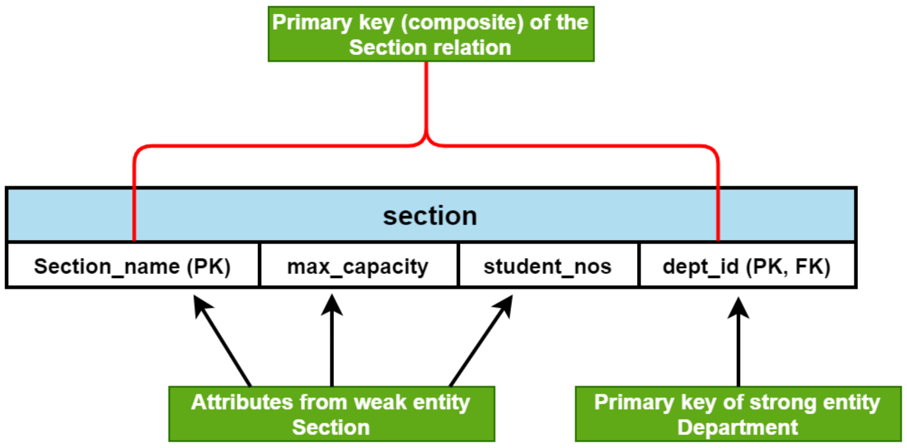

Primary key (composite) of the Section relation
section
Section_name (PK)
max_capacity
student_nos
dept_id (PK, FK)
Attributes from weak entity Section
Primary key of strong entity Department

Figure 2.60 Representation of Weak Entity SECTION (Schema)

In this example, section_name is a partial key attribute of a weak entity type SECTION, and dept_id is the primary key attribute of a strong entity type DEPARTMENT. We will create a foreign key constraint on the SECTION schema referencing the primary key attribute of the DEPARTMENT schema. The combination of dept_id and section_name will form the primary key of Relation SECTION.

2.7.4 Multivalued Attributes

For each multivalued attribute A, create a new Relation R. This Relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the Relation representing the entity type relationship type that has A as an attribute. The primary key of R is the combination of A, and K. For example, phone_no and hobby attributes of STUDENT entity type are multivalued attributes which mean a student can have multiple phone numbers and multiple hobbies.
In this case, we need to create a separate Relation student_phone_no to hold all the phone numbers related to a student. In this new Relation, roll_no from the STUDENT Relation will be included as a foreign key to map different phone numbers of a student to his/her roll_no. The combination of roll_no and phone_no will be the primary key of Relation student_phone_no. We will similarly handle the hobby attribute of the STUDENT entity type.

The schema of Relation STUDENT_PHONE_NO and STUDENT_HOBBY will be:

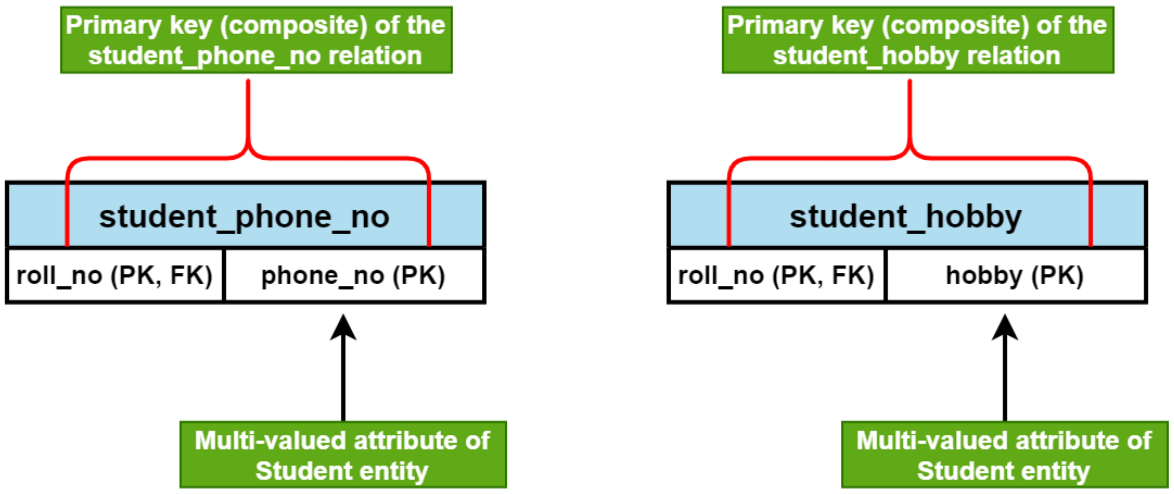

Primary key (composite) of the student_phone_no relation
student_phone_no
roll_no (PK, FK)
phone_no (PK)
Multi-valued attribute of Student entity
Primary key (composite) of the student_hobby relation
student_hobby
roll_no (PK, FK)
hobby (PK)
Multi-valued attribute of Student entity

Figure 2.46 Representation of Multivalued attribute (Schema)

Composite Attributes

While converting composite attributes of an entiry type into a relational representation, we need to create sub-attributes from the composite attribute and define a column for each sub-attribute in the Relation.

For example, address is a composite attribute of STUDENT entity type having sub-attributes: house_no, street_name, city, state, pin_code. We need to create a column for each sub-attribute.

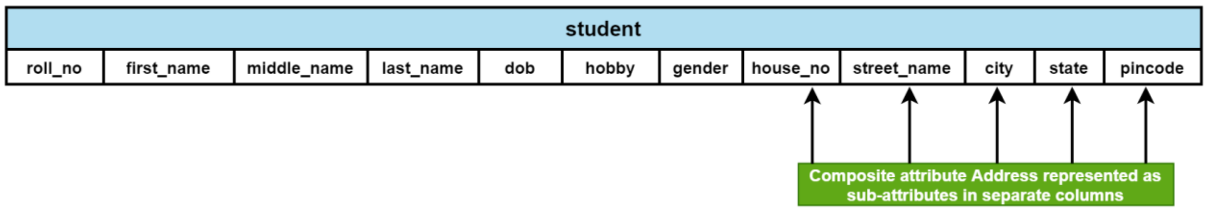


Figure 2.62 Representation of Composite attribute (Schema)

2.7.5 Tabular Representation of Relationship types

For binary 1:1 relationship types:

For each binary 1:1 relationship type R in the ER schema, identify the Relations S and T corresponding to the entity types participating in R.

There are three possible approaches:

1. The foreign key approach,
2. The merged relationship approach, and
3. The cross-reference or relationship relation approach.

1. **Foreign key approach:** Choose one of the Relations say S and include as a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.

   For example, in our case study – Each student has to do one mini-project individually and every available mini-project has to be opted by a student.
   Relationship- Opts
   Cardinality- 1 : 1
   Relation Schema: MINI_PROJECT (miniproj_id, domain, subject, description, roll_no)

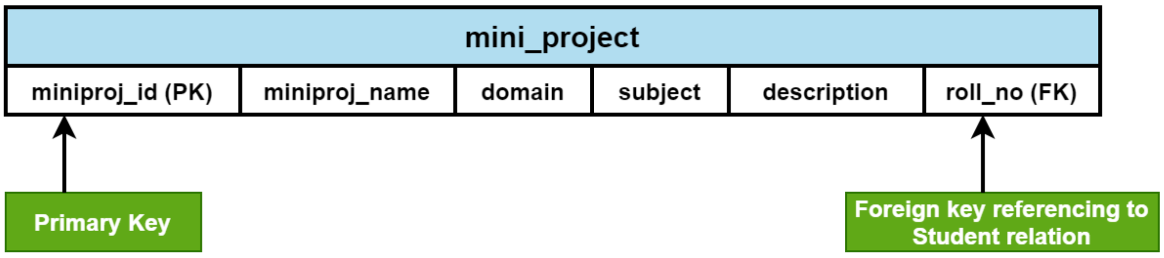


Figure 2.63 Representation of Relation Schema mini_project

   Note that it is possible to include the primary key of S as a foreign key in T instead. Both entity types have total participation in the relationship in this case. So alternate way is that we can add the miniproj_id attribute in the Relation schema of STUDENT as a foreign key attribute referencing the Relation schema of Mini_Project.
2. **Merged relation approach:** An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single Relation. This is possible when both participations are total, as this would indicate that the two Tables will have the exact same number of tuples at all times.
3. **Cross-reference or relationship relation approach:** The third option is to set up a third Relation R to cross-reference the primary keys of the two Relations S and T representing the entity types. As we will see, this approach is required for binary M: N relationships. The Relation R is called a relationship relation (or sometimes a lookup Table) because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T. The Relation R will include the primary key attributes of S and T as foreign keys to S and T. The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R. The drawback is having an extra Relation and requiring extra join operations when combining related tuples from the Tables.

For binary 1: N relationship types:

There are two possible approaches:

1. The foreign key approach and

2. The cross-reference or relationship relation approach.

1. **The foreign key approach:** For each regular binary 1:N relationship type R, identify the Relation S representing the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the Relation T representing the other entity type participating in R; we do this because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of S.

   For example, in our case study – A department can run zero or more courses and each course must belong to only one department.
   Relationship- Runs
   Cardinality- 1: M
   Relation Schema: COURSE (<u>course_id</u>, course_name, course_credit, dept_id)

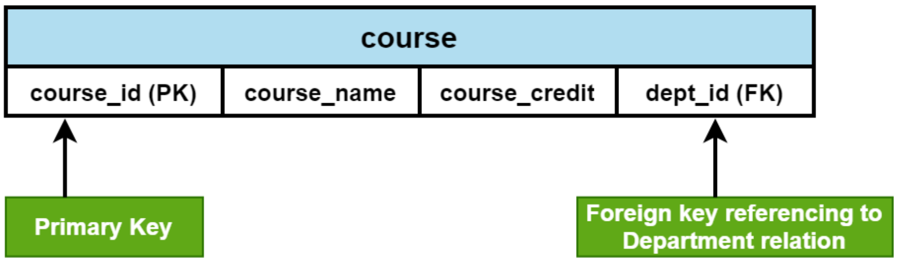


Figure 2.64 Representation of Relation Schema (course)

2. **The relationship relation approach:** An alternative approach is to use the relationship relation (cross-reference) option as the third option for binary 1:1 relationships. We create a separate Relation R whose attributes are the primary keys of S and T, which will also be foreign keys to S and T. The primary key of R is the same as the primary key of S. This option can be used if few tuples in S participate in the relationship to avoid excessive NULL values in the foreign key.

For binary M: N relationship types:

In the traditional relational model with no multivalued attributes, the only option for M:N relationships is the relationship relation (cross-reference) option. For each binary M:N relationship type R, create a new Relation S to represent R. Include as foreign key attributes in S the primary keys of the Relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M: N relationship type (or simple components of composite attributes) as attributes of S. Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating Relations (as we did for 1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate relationship Relation S.

For example, in our case study – A faculty can teach zero or more courses and course can be taught by zero or more faculty.
Relationship- Teaches
Cardinality- M : N
Relation Schema: FACULTY_COURSE (<u>faculty_id, course_id</u>)

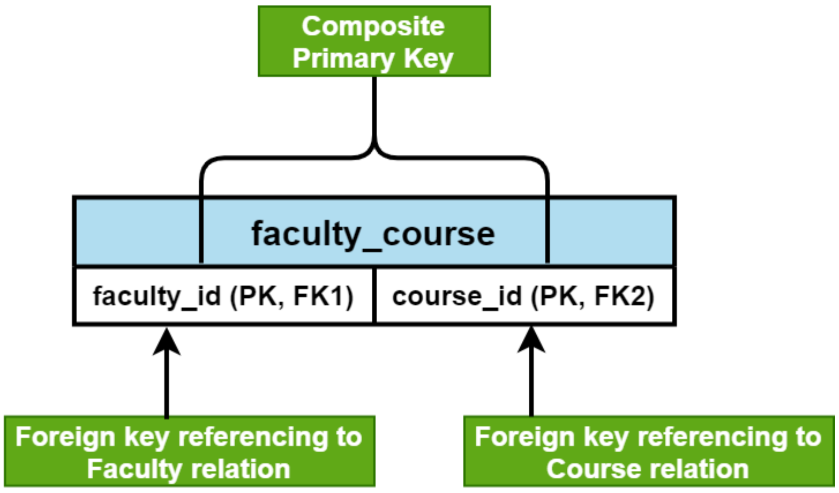

Composite Primary Key
faculty_course
faculty_id (PK, FK1)
course_id (PK, FK2)
Foreign key referencing to Faculty relation
Foreign key referencing to Course relation

Figure 2.65 Representation of Relation Schema (faculty_course)

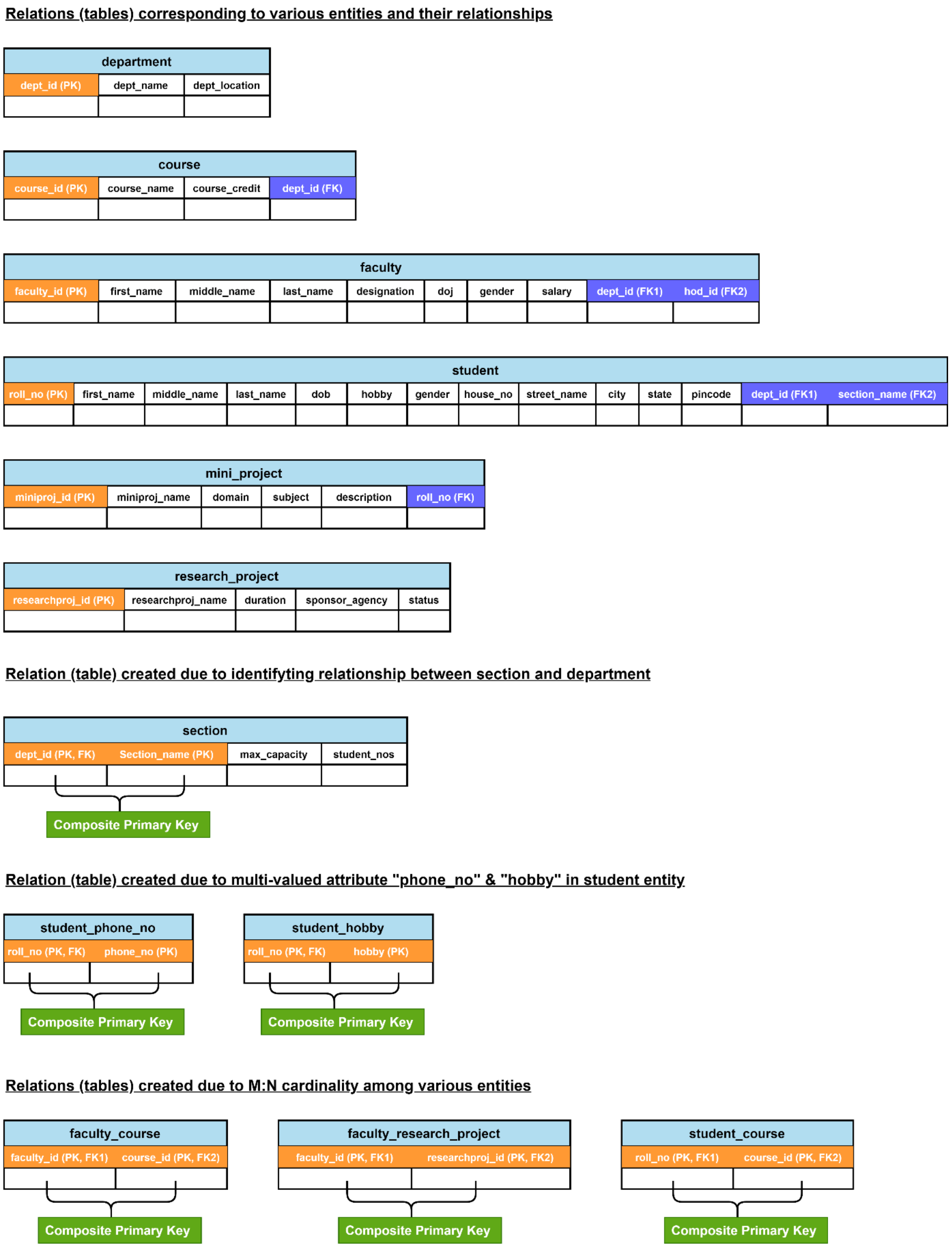
Correspondence between ER and relational model

ER Model	Relational Model
Entity type	"Entity" relation
1:1 or 1:N relationship type	Foreign key
M:N relationship type	"Relationship" relation and two foreign keys
n-ary relationship type	"Relationship" relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Key attribute	Primary key

Figure 2.66 Representation of correspondence between ER and Relational Model

2.8 Complete Relational Model of Case Study

Relations (Tables) of various entities present in case study ER Diagram



Relations (tables) corresponding to various entities and their relationships

department		
dept_id (PK)	dept_name	dept_location

course			
course_id (PK)	course_name	course_credit	dept_id (FK)

faculty									
faculty_id (PK)	first_name	middle_name	last_name	designation	doj	gender	salary	dept_id (FK1)	hod_id (FK2)

student													
roll_no (PK)	first_name	middle_name	last_name	dob	hobby	gender	house_no	street_name	city	state	pincode	dept_id (FK1)	section_name (FK2)

mini_project					
miniproj_id (PK)	miniproj_name	domain	subject	description	roll_no (FK)

research_project				
researchproj_id (PK)	researchproj_name	duration	sponsor_agency	status

Relation (table) created due to identifyting relationship between section and department

section			
dept_id (PK, FK)	Section_name (PK)	max_capacity	student_nos

Composite Primary Key

Relation (table) created due to multi-valued attribute "phone_no" & "hobby" in student entity

student_phone_no	
roll_no (PK, FK)	phone_no (PK)

Composite Primary Key

student_hobby	
roll_no (PK, FK)	hobby (PK)

Composite Primary Key

Relations (tables) created due to M:N cardinality among various entities

faculty_course	
faculty_id (PK, FK1)	course_id (PK, FK2)

Composite Primary Key

faculty_research_project	
faculty_id (PK, FK1)	researchproj_id (PK, FK2)

Composite Primary Key

student_course	
roll_no (PK, FK1)	course_id (PK, FK2)

Composite Primary Key

Figure 2.67 Representation of different Relation Schemas (Tables) in Case Study

Diagram showing various Tables and relationship between them of the case study

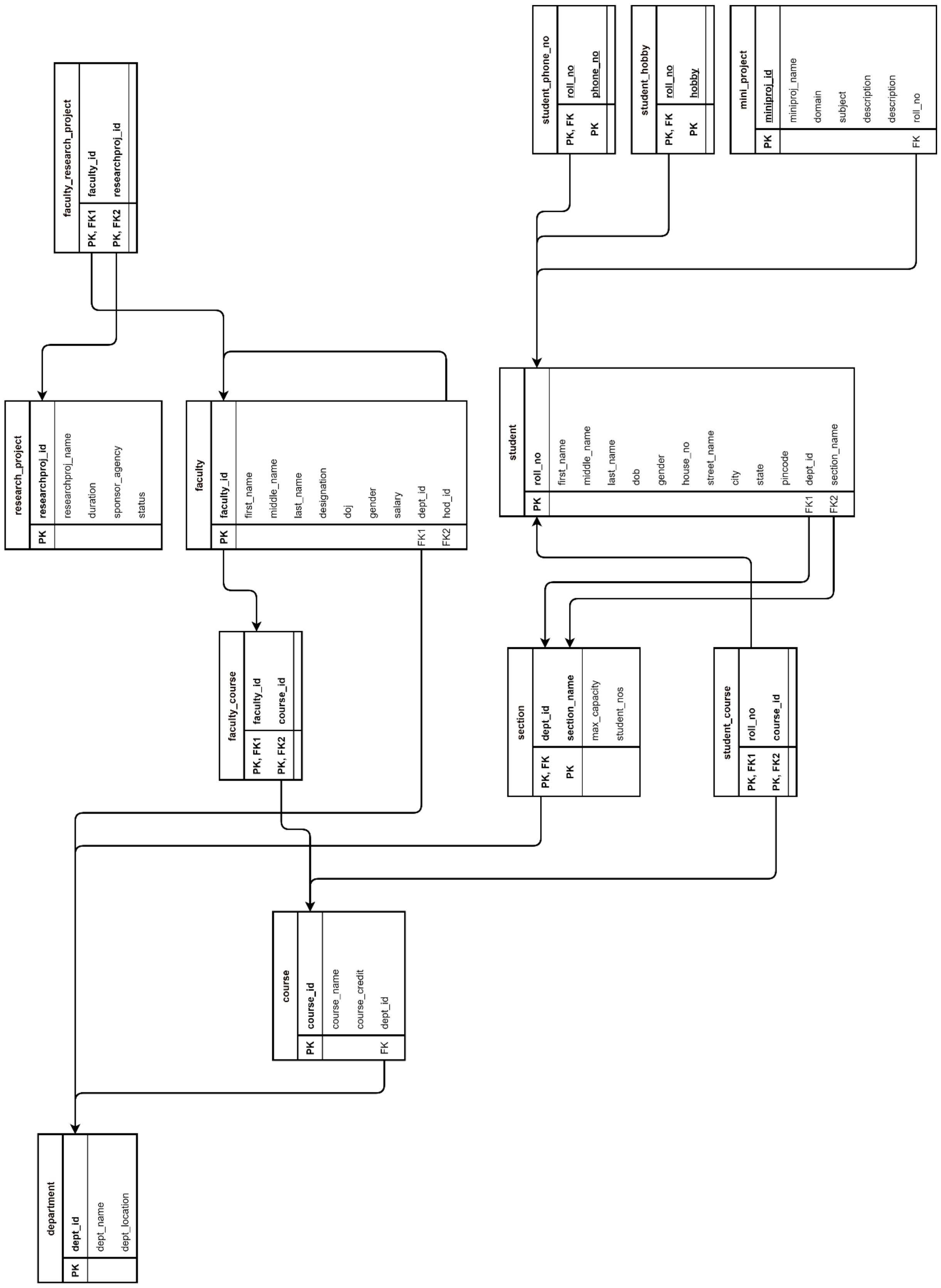


Figure 2.68 Representation of various Tables and the relationship between them

Appendix 1: Various Notations to representation ER Model

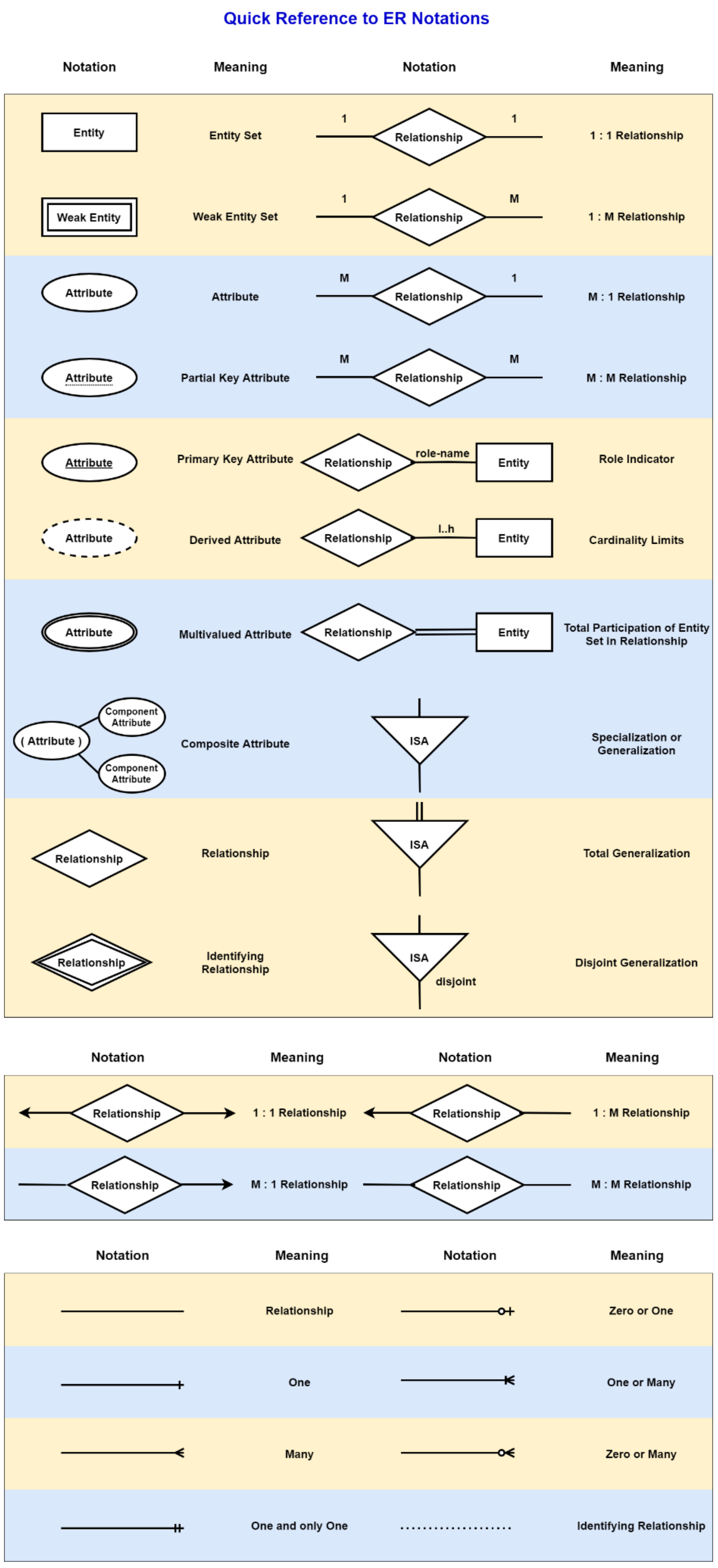


Figure 2.69 Representation of various notations used in ER diagram

Appendix 2: ER Diagram of Case Study in Crow's Foot notation

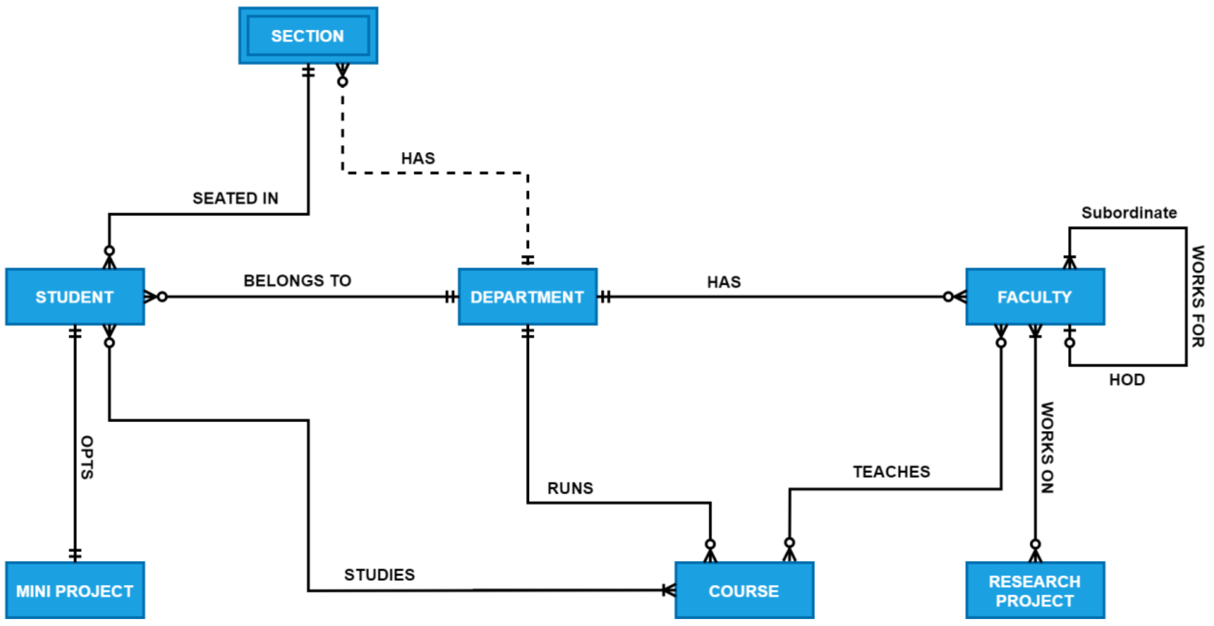


Figure 2.70 Representation of complete ER diagram in Crow's Foot Notation