République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Supérieure d'Économie d'Oran

EDUCATIONAL HANDOUT

From Tables to Queries: Mastering Relational Data with SQL

Courses and Exercises

Prepared by KEMMAR Amina

Academic Year: 2024/2025

SUMMARY

List of figures

CHAPTER 1

INTRODUCTION TO INFORMATION SYSTEMS AND DATABASES

1 INTRODUCTION TO INFORMATION SYSTEMS AND DATABASES

1.1 INTRODUCTION

Today, a company is considered a *system* — but that wasn't always the case. Before the 1970s, it was perceived as a sum of independent entities with defined and separate functions. A company creates value by processing information, especially in the case of service-oriented businesses. Thus, information is all the more valuable as it contributes to achieving the organization's goals.

1.2 NOTION OF A SYSTEM

A **system**, according to Joël de Rosnay, is *"a set of dynamically interacting elements, organized with a purpose."* This definition highlights three key aspects of any system:

1. **A set of elements**: These are the components or parts that make up the system.
2. **Dynamic interactions**: The elements don't function in isolation—they interact, influence, and depend on each other.
3. **A defined purpose**: The system exists to fulfill a specific goal or function.

In other words, a system is more than just a collection of parts—it is an **organized whole**, where each part contributes to a **common objective**, and changes in one part can affect the entire system.

Example: A table manufacturing factory

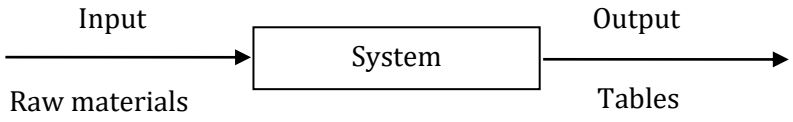


Figure 1 Example of system

Let's apply this definition to a real-world example: **a table manufacturing factory**. This factory is a perfect illustration of a system.

◈ Elements (Components) of the System:

- **Raw materials**: Wood, screws, varnish, metal parts.

- **Machines**: Cutting machines, drilling tools, assembly lines.
- **Human resources**: Workers, technicians, engineers, managers.
- **Information systems**: Production schedules, inventory systems, order tracking software.
- **Energy supply**: Electricity or fuel to power the machines.

◈ Interactions between elements:

- Workers operate machines to cut and shape the wood.
- Assembly processes bring together various parts using tools and screws.
- Managers supervise workflows and make decisions based on production data.
- Inventory systems track material usage and reorder supplies as needed.
- Maintenance teams ensure machines are running properly.

All these components **communicate, exchange, and adapt** to internal or external factors—such as customer demand, supply availability, or equipment performance [1].

◈ Purpose of the System:

The ultimate goal of this system is to **produce tables efficiently and meet customer requirements** in terms of quantity, quality, and delivery time. Every part of the system contributes toward this goal.

Why is this Important?

Viewing an organization, factory, or process as a **system** helps us:

- Understand how different parts are interdependent.
- Identify weaknesses or inefficiencies (e.g., a delay in material delivery affects the entire production).
- Improve performance by optimizing the interactions and flow between elements.
- Adapt to changes by modifying specific components without collapsing the whole.

The organization is seen as a set of means and resources (human, material, financial, technical, etc.) in relation. A company is characterized in the same way as any other system:

- Interacting with its environment and stakeholders,
- Goal-oriented (with aims and objectives),
- Constantly evolving and adapting.

To achieve its strategic objectives defined by top management, the company considers its environment and adjusts its internal organization accordingly.

The business system is composed of three subsystems: the **control (or decision-making) system**, the **operating system**, and the **information system**.

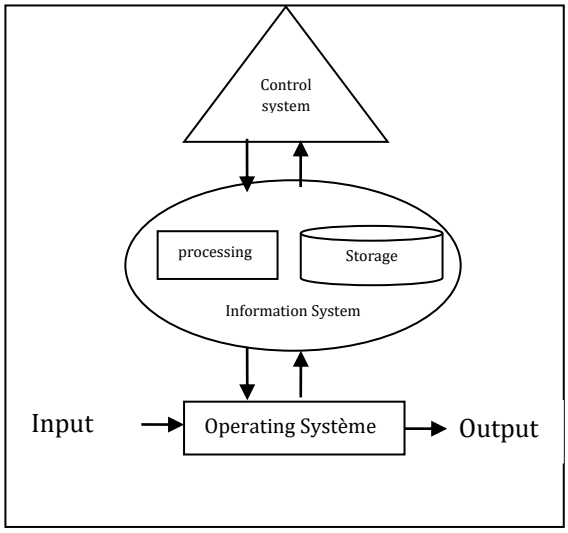


Figure 2 The components of a system

The Control or Decision-Making System:

- Decides and manages the organization of the system,
- Processes the information flowing within the system,
- Implements action plans for the operating system,
- Ensures coherence of actions with the company's strategic goals.

The Operating System:

- Receives decisions from the control system,
- Executes action plans defined by the control system,
- Sends information back to the control system for feedback,
- Includes all operational functions.

1.3 Notion of an Information System (IS)

The *information system (IS)* is now a central element in the functioning of any organization. An IS can be defined as a *set of resources* (people, software, processes, data,

hardware, IT and telecom equipment, etc.) that enable the collection, storage, structuring, modeling, management, manipulation, analysis, transportation, exchange, and distribution of information (texts, images, sounds, video...) within an organization.

To create value, a company must leverage information from both its external environment and its internal operations. The more the information contributes to achieving the company's strategic goals, the more valuable it is.

An Information System (IS) is all the resources (hardware, software, data, procedures, people, ...) structured to acquire, process, store, transmit, and make information (data, text, sounds, images, ...) available within and between organizations." (Robert Reix)

1.4 FUNCTIONS OF AN IS

The IS encompasses the following functions regarding information:

1. Collecting:

This means recording information (on paper or digitally) before processing it. Information can come from:

- **Internal sources:** flows generated by internal company entities (procurement, production, employee management, accounting, sales, etc.), flows formalized by rules and procedures, informal flows (work climate, employee well-being, know-how, etc.).
- **External sources:** flows from external stakeholders (customers, suppliers, the State...), essential environmental data to anticipate changes and adapt operations.

Information is valuable and essential for the company's sustainability — but it's also costly. Any valuable, potentially useful information destined for the IS usually requires human intervention, which is expensive.

2. Storing:

Once collected, information must be stored and preserved. Considerations include:

- Storage must be organized and accessible, so that information can be found easily.

- The storage medium (paper or digital, storage method) must be reliable and long-lasting to ensure information is accessible over time.

3. Processing:

The goal of processing is to make information usable. Processing can be manual (less common) or automatic (via computers). The approach involves choosing a medium and formalizing a processing mode among three types:

- **Centralized:** a single central unit processes the information,
- **Decentralized:** various autonomous entities process and share information,
- **Distributed:** a central site processes data, with terminals used for input/output.

4. Disseminating:

Dissemination is essential for exploiting information. It aims to deliver the right information to the right recipient in a timely manner and in various forms (oral, paper, digital).

The ultimate purpose of the IS is to provide recipients within the company with usable information to support effective decision-making.

1.5 INFORMATION SECURITY

Information security refers to the practice of protecting digital data and information systems from a wide range of threats. These threats can be intentional, such as **malicious attacks**, or unintentional, such as accidental data loss or system failure. The goal is to ensure the **confidentiality**, **integrity**, and **availability** of information—often abbreviated as the **CIA triad**.

A) **Protection from Malicious Attacks and External Threats**

One of the core aspects of information security is defending systems against **external threats**—such as hackers, malware, and cybercriminals—who aim to access, alter, or destroy sensitive information. Organizations implement a variety of **technical safeguards** to counter these risks:

- **Firewalls**: Act as barriers between internal networks and the outside world, filtering incoming and outgoing traffic based on predefined security rules.

- **Filtering routers**: Inspect and control data packets moving through the network to prevent unauthorized access or data leaks.
- **Antivirus software**: Scans systems for known viruses, malware, and spyware, and helps remove or quarantine malicious files before they cause harm.
- **Intrusion Detection Systems (IDS)**: Monitor network traffic for suspicious behavior or signs of unauthorized access, triggering alerts when threats are detected.

Together, these tools help organizations **detect, prevent, and respond** to cyberattacks in real time.

B) Protection from Data Loss

Information security is not just about keeping intruders out—it's also about making sure that valuable data isn't lost due to accidents, disasters, or technical failures. This is achieved through:

- **Regular data backups**: Making copies of critical data at regular intervals so that it can be restored in case of loss.
- **Disaster recovery plans**: Establishing procedures to recover data and resume operations quickly after major disruptions, such as power outages, natural disasters, or hardware failures.
- **Redundant systems**: Using duplicate servers or storage devices that can take over automatically if the main system fails.

These measures ensure that information remains **available and recoverable**, even in unexpected situations.

C) People and Policies Matter Too

Technology alone isn't enough. Effective information security also depends on:

- **Employee training** (e.g., avoiding phishing emails, using strong passwords),
- **Access control policies** (e.g., only authorized users can access certain files),

- **Compliance with legal and regulatory standards** (e.g., GDPR, HIPAA).

Information security is a comprehensive field that includes both **technical defenses** (like firewalls and antivirus) and **organizational practices** (like backups and security policies). It protects data from being **stolen, damaged, or lost**, ensuring that information remains safe, accurate, and accessible when needed.

1.6 DATA STORAGE

Computerizing the Information System (IS) requires the implementation of hardware and software supports for data storage. Information is digitized and stored as files.

Let's take a simple example. If you develop a software application, a video game, or even a website later on, you will (at a fairly advanced stage) face a problem: how to save all the data used by the application? This is where the well-known **database (DB)** comes into play. A **DB** stores information in a way similar to a file. A database is indeed a file, but saved in a special format.

1.7 CONCEPT OF A DATABASE (DB)

A database is a structured set of data representing real-world information, stored on media accessible by a computer, in order to simultaneously serve multiple users (data sharing), selectively (confidentiality), and in a timely manner (performance).

1.8 DATABASE MANAGEMENT SYSTEM (DBMS)

Just like in libraries where people are responsible for sorting, organizing, and updating information about books, there are software tools that do the same job with our databases. These are called **Database Management Systems**, or **DBMS** for short.

A **DBMS** (Database Management System) is software that allows users to define, create, and update a database, as well as control access to it. It acts as an interface between the database and the user. It provides the means to define, control, store, manipulate, and process data while ensuring the security, integrity, and confidentiality that are essential in a multi-user environment [2].

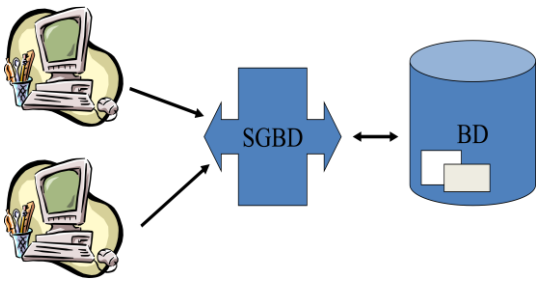


Figure 3 Interaction entre utilisateur et BDD

We can tell that:

- A **database** is where the data lives.
- A **DBMS** is the software that helps you take care of that data.

We can think of it like this:

Database = Data itself
DBMS = The manager that keeps everything running smoothly

1.9 DATA STRUCTURING – DATA REPRESENTATION MODEL

In a **Database Management System (DBMS)**, data is not stored randomly. Instead, it is **organized according to a specific framework** known as a **data model**. A data model defines **how data is structured**, **how relationships between data are represented**, and **how data can be manipulated and accessed**.

Just like blueprints guide the construction of buildings, data models provide a logical plan for designing, storing, and managing data in a consistent and efficient way.

There are several types of data models, each offering different ways to organize and relate data depending on the needs of the system or application. The most commonly used data models include:

1. Hierarchical Data Model

This is one of the earliest data models, where data is organized in a **tree-like structure**. Each record has a **parent-child relationship**, meaning each child has only one parent, but a parent can have multiple children.

- **Example**: Think of an organization chart where the CEO is at the top, followed by managers, and then employees. The structure is strictly one-to-many.
- **Pros**: Simple and easy to understand when the data has a natural hierarchy.
- **Cons**: It lacks flexibility. If relationships change or become more complex, the model becomes difficult to maintain.

2. Network Data Model

The network model is an improvement over the hierarchical model. In this model, records are organized in a graph, allowing **many-to-many relationships**.

- **Example**: A student can enroll in multiple courses, and each course can have multiple students. This cannot be easily represented in the hierarchical model but is natural in the network model.
- **Pros**: More flexible than the hierarchical model, especially for complex relationships.
- **Cons**: It can become complicated to design and manage due to its intricate structure.

3. Relational Data Model

This is the most widely used model today. In the **relational model**, data is organized into **tables (also called relations)**. Each table contains **rows** (records) and **columns** (fields), and relationships between tables are established using **keys** (e.g., primary keys and foreign keys).

- **Example**: A Customers table and an Orders table can be linked by a CustomerID field. This makes it easy to find all orders placed by a specific customer.
- **Pros**: Simple, flexible, and supported by powerful query languages like SQL. Ideal for most business applications.
- **Cons**: May be less efficient for very complex, deeply interconnected data.

4. Object-Oriented Data Model

This model integrates the principles of object-oriented programming. Data is stored in **objects**, which include both the **data (attributes)** and the **methods (functions)** that operate on that data.

- **Example**: A Car object may contain properties like color, brand, and speed, as well as methods like start() or accelerate().
- **Pros**: Good for applications that require a tight coupling between data and operations, like in software development.

- **Cons**: Less common in traditional business environments; more complex to implement and understand.

5. Document and NoSQL Models (Modern Models)

With the rise of big data and web applications, new types of models—often referred to as **NoSQL**—have become popular. These models are more flexible and are designed for unstructured or semi-structured data.

- **Examples**:
    - **Document model** (e.g., MongoDB): Stores data in JSON-like documents.
    - **Key-value stores** (e.g., Redis): Store data as a collection of key-value pairs.
    - **Graph databases** (e.g., Neo4j): Ideal for data with many relationships, like social networks.
- **Pros**: High performance, flexible structure, ideal for big data and distributed systems.
- **Cons**: Lack the strict structure and standardization of relational models; not always suitable for traditional applications.

In Summary, a **data model** is essential to how a DBMS stores and manages data. Each model:

- Defines the logical structure of the database,
- Specifies how data is related,
- Influences how efficiently data can be accessed and modified.

Choosing the right data model depends on the **type of data**, the **complexity of relationships**, and the **specific needs of the application**. While the **relational model** remains dominant in many domains, modern applications are increasingly making use of more flexible models like **NoSQL** to handle the growing complexity and volume of data.

The commonly used models are summarized in the following table:

Model	Structure
Hierarchical	Trees (1960s)
Network	Graphs (early 1970s)
Relational	**Relations (early 1980s). Examples of RDBMS: MySQL, ORACLE, SQL Server, Access.**
Object-oriented	Objects

The most common data model is the relational data model, which uses a construct called a "relation," in the mathematical sense of a set.

1.10 DATABASE CONSTRUCTION APPROACH

To build a database, you need to:

1. Construct a conceptual schema, modeled in the form of entities and associations (E/A model),
2. Transform the E/A schema into a relational schema,
3. Implement it via a DBMS such as ACCESS (Inisan) (Sparfel).

The conceptual data model (CDM) specifies the structure of the database independently of the DBMS used for data exploitation, while being the most faithful representation possible of the organizational reality as perceived through the data. We mention the Entity/Association model, which will be the subject of the following chapter.

CHAPTER 2

THE ENTITY/RELATIONSHIP (E/R) MODEL

2 THE ENTITY/RELATIONSHIP (E/R) MODEL

2.1 INTRODUCTION

The modeling of a problem, i.e., the transition from the real world to its computer representation, is defined in several steps to integrate it into a DBMS (Database Management System) and allow data manipulation using the SQL language. The first level of modeling, called conceptual, consists of an analysis phase of the real problem. This phase is quite delicate and helps define the data to be used, their evolution over time, and the relationships between them. Among these models, the E/R model represents a graphical formalism for data modeling [3].

2.2 BASIC CONCEPTS

A. Attribute:

An attribute is a property that describes an object in the universe (e.g., color of a car, name of a student). There are several types of attributes:

- ***Simple attribute**: Not divisible (e.g., age).*
- ***Composite attribute**: Can be subdivided into simpler attributes in a hierarchical form (e.g., mailing address = street + postal code + city + country).*
- ***Derived attribute**: Its value is calculated (e.g., total price including tax derived from the price excluding tax).*
- ***Null value**: When the value of the attribute is unknown. For example, if a customer's address is unknown, the value NULL is assigned.*
- ***Attribute domain**: The set of values an attribute can take (e.g., if the age attribute can range from 0 to 100, then its domain is [0,100]; if a grade ranges from 0 to 20, its domain is [0,20]).*

B. Entity:

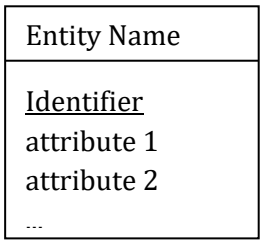
***Entity** = A concrete or abstract object from the universe of discourse (e.g., a car, a product).*

Entity type = A grouping of entities of the same nature that share common semantics and properties. People, books, and cars are examples of entity types. In the case of a person, for instance, associated information (or properties) such as first name and last name do not change in nature from one person to another. To simplify things, the term "entity" is often used to refer to an entity type.

Entity occurrence = A set of attribute values that characterize a specific entity.

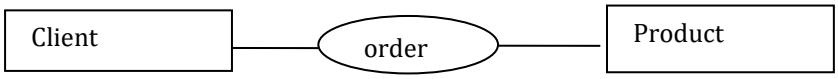
Entity identifiers = An attribute that uniquely identifies each occurrence of an entity type.

2.3 GRAPHICAL REPRESENTATION



2.4 ASSOCIATIONS AND CARDINALITIES

An **association** is used to link one or more entities. For example, *customers order products*.



The **cardinality** of an association defines how entities are related. It is defined by two numbers *(min, max)* representing the minimum and maximum number of times an entity can participate in an association.

min: Answers the question

— *How many times at least is an entity of A linked to an entity of B?*

max: Answers the question

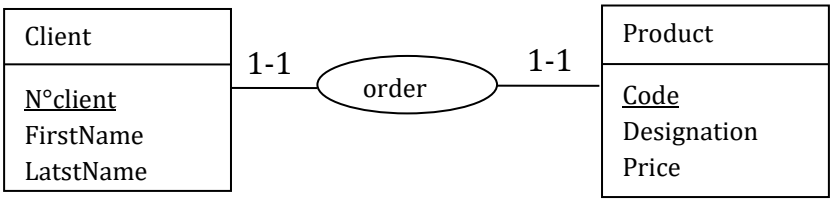
— *How many times at most is an entity of A linked to an entity of B?*

These questions must be asked in **both directions**: from A to B, and then from B to A.

A. Types of association:

- *Association 1-1*: The association described is a **One-to-One (1:1)** relationship, where:
  - ✓ **One client** place **one order** for a **single product**.
  - ✓ Each **product** is ordered by **one client** only.

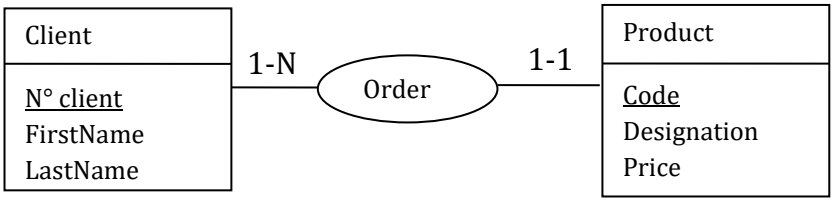
This can be represented as a **1:1** relationship between the "Client" and "Product" entities, where each entity in one set is linked to exactly one entity in the other set.



- *Association 0-N or 1-N*: The association you describe is a **One-to-Many (1:N)** relationship, where:
  - ✓ **One client** can order **multiple products**.
  - ✓ Each **product** is ordered by **only one client**.

This means that a client can place multiple orders (each for potentially different products), but each product is associated with exactly one client. It is represented as **1:N** (One-to-Many), with "client" being the "1" side and "product" being the "N" side.

In a graphical representation, you would show an arrow pointing from "Client" to "Product" with a "1" on the "Client" side and an "N" on the "Product" side to indicate this relationship.



Note: The **"one-to-many" (1:N) cardinality** can also be expressed as **"zero-to-many" (0:N)** when a client exists but may not necessarily order a product.

In this case, it means that:

- A **client** may or may not place an order for a product (so, **0** products or **many** products).
- A **product** is ordered by only one client.

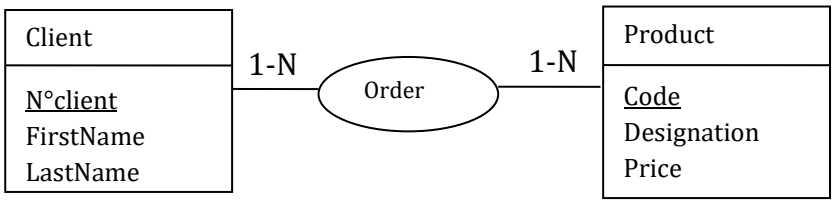
This can be represented as **0:N**, where the "0" indicates that a client might not place any orders, and the "N" indicates that if a client does order, they can order multiple products.

- *Association M-N*: The relationship you describe is a **Many-to-Many (M:N)** association, where:
    - ✓ **One client** can order **multiple products**.
    - ✓ **One product** can be ordered by **multiple clients**.

This is a **M:N** relationship because each entity in both sets (clients and products) can be associated with multiple entities in the other set.

In a graphical representation, you would show an association between "Client" and "Product" with "M" on both sides to indicate that multiple clients can order multiple products, and vice versa.

To implement this in a relational database, you'd typically introduce a **junction table** (also called a **bridge table** or **associative entity**) to handle the relationship between clients and products. This table would have foreign keys pointing to both the "Client" and "Product" entities.



B. Association Attributes:

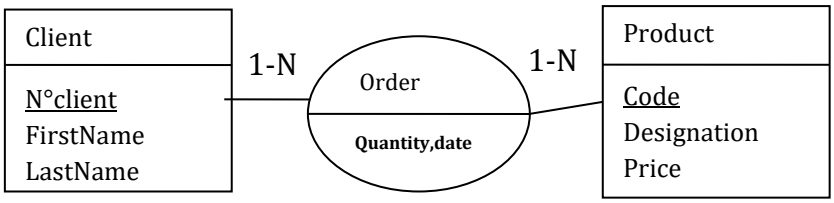
In an Entity-Relationship (E/R) model, **attributes of an association** are properties or characteristics that are associated with a relationship between two or more entities, rather than just the entities themselves.

For example, in the **Many-to-Many (M:N)** relationship between **Client** and **Product**, an association might have attributes like:

- **Quantity**: The number of units of a product ordered by a client.
- **Order Date**: The date on which the product was ordered by the client.

- **Delivery Status**: The status of the delivery for the ordered product (e.g., pending, shipped, delivered).

These attributes provide additional information about the relationship itself. To represent this in a relational database, the attributes would typically be placed in the **junction table** (or associative entity) between the entities involved in the association.



2.5 STEPS TO FOLLOW TO PRODUCE AN E/R DIAGRAM

Here are the typical steps to follow to create an Entity-Relationship (E/R) diagram:

1. **Identify the Entities**: Identify the main objects or concepts in the system. These will be represented as entities in the E/R diagram. For example, if you are modeling a store system, you might have entities like "Customer," "Product," and "Order."

2. **Define the Attributes of Each Entity**: For each identified entity, list its attributes. These attributes describe the properties of the entity. For example, the "Customer" entity might have attributes like "Customer ID," "Name," and "Email."

3. **Identify Relationships Between Entities**: Determine how the entities are related to one another. A relationship is a link between two or more entities. For example, "Customers" place "Orders" and "Orders" contain "Products."

4. **Determine the Cardinalities**: For each relationship, define the cardinality (min, max). This will show how many times one entity can be associated with another. For example, a **one-to-many** relationship might indicate that one "Customer" can place many "Orders," but each "Order" is linked to exactly one "Customer."

5. **Add Relationship Attributes (if needed)**: If the relationship itself needs further attributes (like "Order Date" or "Quantity"), these are defined as **attributes of**

the association. In the case of a **many-to-many** relationship, you might use an associative entity to represent these attributes.

6. **Draw the Diagram**: Use a graphical tool or diagramming software to visually represent the entities, relationships, attributes, and cardinalities. Draw rectangles for entities, ovals for attributes, diamonds for relationships, and connect them with lines to show how they are related.

7. **Refine the Diagram**: Review and refine the E/R diagram to ensure it accurately represents the system and its requirements. Make sure that all relevant relationships are captured and that the cardinalities are correct.

8. **Validate the Diagram**: Finally, validate the E/R diagram with stakeholders (e.g., users, developers, or domain experts) to ensure that it aligns with the real-world requirements and business rules.

This process will help create a comprehensive and accurate **Entity-Relationship diagram** that serves as the foundation for database design and development.

A series of exercises with solutions is provided in (emmanuel.coquery).

Example 1: How to represent data on student enrollments for courses, given that:

- 1 teacher teaches one or more subjects to students.
- 1 student takes one or more subjects, each with a teacher.

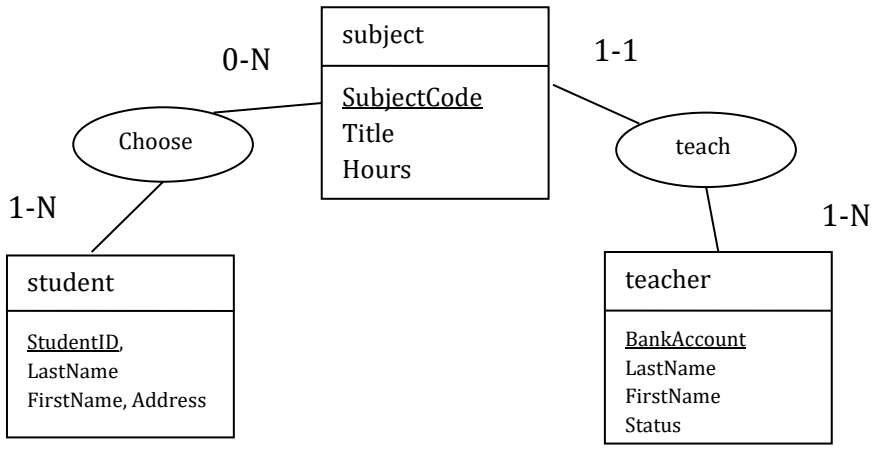
Steps to follow:

1) **Identify Entities and attributes**

   Teacher (LastName, FirstName, Status, BankAccount)
   Student (StudentID, LastName, FirstName, Address)
   Subject (SubjectCode, Title, Hours)

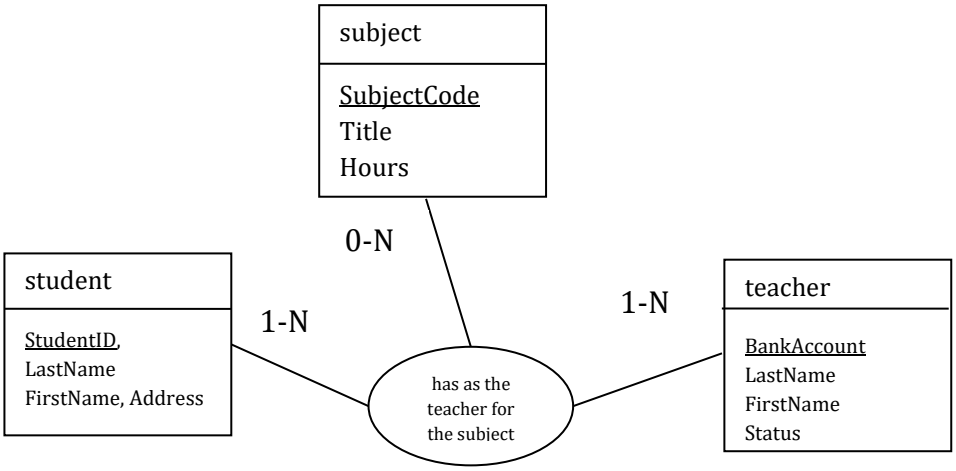
2) **Identify associations and cardinalities**

Proposition 1: Binary Associations



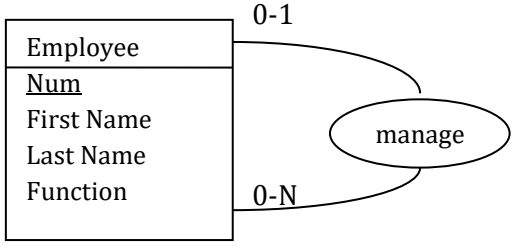
Proposition 2: Using a ternary association

In the context of databases or entity-relationship (ER) modeling, a **ternary association** is a relationship that involves **three entities** simultaneously. It is used when a relationship cannot be properly represented using only binary associations between pairs of entities.



Example 2 (reflexive association):

In database or ER modeling, a reflexive association is a relationship where an entity is related to itself. This means that instances of the same entity type are related to each other.



In this example, every employee is managed by another employee (except the general manager), and one employee can manage several employees. An employee is characterized by a unique number, a last name, a first name, and their position.

Note: For more examples, I invite the reader to consult (Audibert).

2.6 EXERCISES

2.6.1 EXERCISE 1:

The information system of a university uses the following data:

– For each student: their student ID number, last name, first name, and address (street, number, postal code, city).

– For each course: the mnemonic code, the title, and a short summary.
– For each professor: their employee ID number, title, last name, first name, and address.

Furthermore, the system must keep track of the courses each student takes, the professor in charge of each course, the program (major) chosen by each student, and for each program (identified by its name and code), the professor who manages it. It should be noted that each student can take a maximum of 10 courses, and a course is taught by only one professor who can be responsible for only one program.

Question: Propose an entity-relationship (ER) model for this system.

2.6.2 EXERCISE 2:

A **library** system needs to be designed to manage books, authors, and members. The system should track the books borrowed by members, as well as details about authors and their books.

Here's a summary of the entities and their relationships:

1. **Books**
   Each book has a **title**, an **ISBN**, and a **publication year**.
   A book can have **multiple authors**.
   Each book can be **borrowed by members**.

2. **Authors**

   Each author has a **name** and a **birthdate**.

   An author can write **multiple books**.

3. **Members**

   Each member has a **membership number**, a **name**, and an **address**.

   A member can **borrow multiple books** but only one copy of each book at a time.

4. **Borrows**

   This is a relationship between **members** and **books**.

   Each borrow record includes:

   - **borrow date**
   - **return date**

Tasks:

1. **Identify the entities** in the system and list their attributes.
2. **Identify the relationships** between entities (e.g., "borrows", "writes") and describe their cardinality (one-to-many, many-to-many).
3. **Draw the ER diagram** for the system, including:
   - Entities with attributes.
   - Relationships with cardinalities.
   - Identify primary and foreign keys.

2.6.3 Exercise 3: Nursery Management System

Scenario:

The system is designed to manage the children, caregivers, and activities in a nursery. The nursery should track children's personal information, their attendance, the caregivers assigned to each child, and the activities they participate in.

Here's a summary of the entities and their relationships:

1. **Children**

   Each child has a **child ID**, **name**, **date of birth**, and **guardian's contact information**.

   A child can participate in **multiple activities** and can have **multiple attendance records**.

   Each child is assigned to a **caregiver**.

2. **Caregivers**

   Each caregiver has a **caregiver ID**, **name**, **age**, **qualification**, and **shift schedule**.

   A caregiver is assigned to **multiple children** and can be responsible for **multiple activities**.

3. **Activities**

   Each activity has an **activity ID**, **name**, and **duration** (start time and end time).

   A child can participate in **multiple activities**, and each activity can have **multiple children**.

4. **Attendance**

   Each attendance record has an **attendance ID**, **date**, and **status** (present/absent).

   A child can have multiple **attendance records** for each day they attend the nursery.

5. **Guardians**

   Each guardian has a **guardian ID**, **name**, **relationship to child**, and **contact information**.

   A guardian can have **one or more children** in the nursery.

Tasks:

1. **Identify the entities** in the system and list their attributes.
2. **Identify the relationships** between entities (e.g., "participates in", "assigned to", "records attendance") and describe their cardinality (one-to-many, many-to-many).
3. **Draw the ER diagram** for the system, including:
   - Entities with attributes.
   - Relationships with cardinalities.
   - Identify primary and foreign keys.

CHAPTER 3

THE RELATIONAL MODEL

3 THE RELATIONAL MODEL

3.1 INTRODUCTION

The relational model was introduced and formalized by **Edgar F. Codd** in **1970** in a groundbreaking paper titled *"A Relational Model of Data for Large Shared Data Banks."* This model marked a major shift in the way databases were conceived and structured, laying the foundation for modern database systems.

At its core, the relational model organizes data into **tables**, also called **relations**, where each table is composed of **rows** (tuples) and **columns** (attributes) (Jouve). Each table represents a specific entity or relationship, and each row within a table corresponds to a unique record. This tabular representation allows for a clear, simple, and consistent way to model real-world data.

One of the most significant aspects of the relational model is the separation between the **logical** and **physical** levels of data organization. The model defines the **logical structure** of data—how it is presented to users—without prescribing how the data must be stored on disk. This abstraction allows database management systems (DBMS) to optimize storage and access internally, using any physical storage technique, while maintaining a consistent and user-friendly logical interface.

The relational model also provides a strong theoretical foundation based on mathematical concepts, particularly **set theory** and **first-order predicate logic**. This makes it possible to define powerful query languages, such as **SQL (Structured Query Language)**, which allow users to manipulate and retrieve data efficiently and declaratively.

Because of its simplicity, flexibility, and robustness, the relational model quickly became the dominant paradigm for database systems and remains the standard today in both academic and commercial environments.

3.2 BASIC CONCEPTS

Attribute: An attribute is a name that describes a piece of information stored in a database (it is the name given to a column in a relation, which takes its values from a domain).
Examples: a person's age, their name, their social security number.

Domain: The domain of an attribute is the set—finite or infinite—of its possible values.
Examples: the attribute *social security number* has as its domain the set of all fifteen-digit combinations; the attribute *price* has as its domain the set of positive integers.

Relation: A relation (denoted *R*) is a subset of the Cartesian product of *n* attribute domains ($n > 0$): $R \subset D_1 \times D_2 \times \ldots \times D_n$
$D_1, D_2, \ldots, D_n$ are the domains of the attributes of *R*, and *n* represents the degree of *R*.
A relation is represented as a two-dimensional table in which the *n* attributes correspond to the titles of the *n* columns.
Relation Schema: A relation schema specifies the name of the relation as well as the list of its attributes.
Example of a relation and its schema:
Let the following domains be defined:

- D(Num_Inscription) = [1..500]
- D(LastName) = {Chadli, Tabeti}
- D(FirstName) = {Fatima, Mohammed, Ibrahim}
- D(Date_Birth) = {Dates between 01/01/1990 and 31/12/2001}

We define the relation STUDENT as follows:

STUDENT ⊂ D(Num_Inscription) × D(LastName) × D(FirstName) × D(Date_Birth)

This relation is described by the following relation schema:
STUDENT(Num_Inscription, LastName, FirstName, Date_Birth)
Its representation in the form of a table is:

Student			
Num_Inscription	**LastName**	**FirstName**	**Date_Birth**
200	Tabti	Fatima	25/04/2000
201	Chadli	Mohammed	10/01/1999
203	Chadli	Ibrahim	12/06/1999

Degree of a relation: The degree of a relation is the number of its attributes. ***For example,*** the degree of the STUDENT relation is 4.

Occurrence: An occurrence, or n-tuple or tuple, is an element of the set represented by a relation. In other words, an occurrence is a row in the table that represents the relation. ***For example***, the row (201, Chadli, Mohammed, 10/01/1999) represents a tuple for the ETUDIANT relation.

Candidate key: A candidate key of a relation is an attribute or a set of attributes that uniquely identifies each occurrence in that relation. Every relation must have at least one candidate key. ***For example***, the attribute Num_Inscription is unique (you cannot have two students with the same registration number), so it is considered a candidate key.

Primary key: The primary key of a relation is one of its candidate keys. To indicate the primary key, it is typically underlined.

Foreign key: A foreign key in a relation is made up of one or more attributes that form a primary key in another relation.

Relational schema: A relational schema is made up of the set of relation schemas, including references to foreign keys.

Relational database: A relational database is composed of the set of n-tuples from the different relations in the relational schema.

A summary example:

Let's consider the EMPLOYEE relation with the following schema:
Employee(Emp_ID, First_Name, Last_Name, Department_ID, Salary)
This relation has 5 attributes:

- Emp_ID (Employee ID)
- First_Name (Employee's first name)
- Last_Name (Employee's last name)
- Department_ID (ID of the department the employee works in)
- Salary (Employee's salary)

Therefore, the degree of the relation is 5 because there are 5 attributes in the relation.

Here is a possible table for **EMPLOYEE**:

Emp_ID	First_Name	Last_Name	Department_ID	Salary
101	Tabti	Mohammed	1	50000
102	Fkir	Asmaa	2	60000

- The row (101, Alice, Johnson, 1, 50000) is an **occurrence** (or **tuple**) in the relation **EMPLOYEE**. It represents the data for a single employee, Alice.
- In the **EMPLOYEE** relation, the **Emp_ID** (Employee ID) is a **candidate key** because it uniquely identifies each employee. No two employees can have the same **Emp_ID**. Therefore, **Emp_ID** is a candidate key.
- Alternatively, the combination of **First_Name** and **Last_Name** might also be unique in some cases, but it's less practical. **Emp_ID** is the best choice for uniqueness.
- From the previous example, **Emp_ID** is the **primary key** of the **EMPLOYEE** relation because it uniquely identifies each employee. The schema for **EMPLOYEE** would look like this: **Employee(<u>Emp_ID</u>, First_Name, Last_Name, Department_ID, Salary)** where **Emp_ID** is underlined to indicate it's the primary key.
- Now, consider the **DEPARTMENT** relation, which stores information about departments in the company. It might look like this:

Department_ID	Department_Name
1	Sales
2	Marketing

The **EMPLOYEE** relation has a **Department_ID** attribute, which is a **foreign key** because it refers to the **Department_ID** in the **DEPARTMENT** relation. This creates a connection between the two tables.

The relational schema for the EMPLOYEE relation would be:

Employee(Emp_ID, First_Name, Last_Name, Department_ID, Salary)

The DEPARTMENT relation schema would be:

Department(Department_ID, Department_Name)

And the EMPLOYEE relation would reference Department_ID as a foreign key.

In our case, the **relational database** consists of the following two relations:

1. **EMPLOYEE**:
   Employee(Emp_ID, First_Name, Last_Name, Department_ID, Salary)
2. **DEPARTMENT**:
   Department(Department_ID, Department_Name)

The **Department_ID** in **EMPLOYEE** acts as a **foreign key** linking it to the **DEPARTMENT** table. Together, these two relations make up a relational database.

3.3 INTEGRITY CONSTRAINTS

In the context of databases, **integrity constraints** are essential rules designed to ensure the **accuracy, consistency, and reliability** of data within the database, especially when the database undergoes updates. These updates include operations like **insertion**, **modification**, or **deletion** of data. The purpose of integrity constraints is to preserve the correctness of the data, even as changes are made. Without these constraints, there is a risk of introducing inconsistencies, such as duplicate records, incorrect data relationships, or violating the inherent rules of the database's design.

Integrity constraints are particularly crucial in **relational databases**, where data is stored in **tables** (or relations), and relationships between data across different tables must be managed effectively. These constraints must be checked for every **tuple** (or record) that makes up the database, ensuring that each record adheres to the established rules.

In a relational database, integrity constraints fall into several categories, each serving a distinct role in maintaining data integrity. The following three categories of **structural integrity constraints** are particularly important:

1. **Entity Integrity**: This ensures that every row (tuple) in a relation has a unique identifier, typically through the use of **primary keys**.
2. **Referential Integrity**: This ensures that relationships between tables are valid and that foreign keys in one table correspond to valid entries in another table.
3. **Domain Integrity**: This guarantees that the data in each column of a table adheres to the defined domain, meaning that each value is of the correct type and falls within the permissible range of values.

By enforcing these constraints, databases maintain their reliability and consistency over time, preventing erroneous data from compromising the integrity of the entire system.

3.3.1 Example on Entity Integrity:

The **primary key** in the **EMPLOYEE** relation is **Emp_ID** (Employee ID). Each **Emp_ID** must be unique, meaning that no two employees can have the same **Emp_ID**.

EMPLOYEE Table (with Entity Integrity):

Emp_ID (PK)	First_Name	Last_Name	Department_ID (FK)	Salary
101	Tabti	Mohammed	1	50000
102	Fkir	Asmaa	2	60000
103	Allami	Omar	1	55000

If we try to insert a row with the same **Emp_ID** (e.g., **Emp_ID = 101** for another employee), the **entity integrity constraint** would be violated because **Emp_ID** must be unique across all records.

Invalid Example:

Emp_ID (PK)	First_Name	Last_Name	Department_ID (FK)	Salary
101	Fares	Ali	2	45000

3.3.2 EXAMPLE ON REFERENTIAL INTEGRITY:

Referential integrity ensures that foreign keys in a table correspond to valid entries in another table. In our case, the **Department_ID** in the **EMPLOYEE** table is a foreign key that references the **Department_ID** in the **DEPARTMENT** table. This constraint guarantees that every employee is assigned to a valid department.

- **Example in the EMPLOYEE and DEPARTMENT tables**:

DEPARTMENT Table (with primary key **Department_ID**):

Department_ID (PK)	Department_Name
1	Sales
2	Marketing

EMPLOYEE Table (with foreign key **Department_ID**):

Emp_ID (PK)	First_Name	Last_Name	Department_ID (FK)	Salary
101	Tabti	Mohammed	1	50000
102	Fkir	Asmaa	2	60000
103	Allami	Omar	1	55000

Referential Integrity Rule:

- Every **Department_ID** in the **EMPLOYEE** table must exist in the **DEPARTMENT** table.
- If we attempt to insert an employee with a **Department_ID** that doesn't exist in the **DEPARTMENT** table (for example, **Department_ID = 3**), it will violate the **referential integrity constraint**.

Invalid Example:

Emp_ID (PK)	First_Name	Last_Name	Department_ID (FK)	Salary
104	Moumni	Nassim	3	70000
(This would violate referential integrity because Department_ID = 3 does not exist in the DEPARTMENT table.)				

3.3.3 EXAMPLE ON DOMAIN INTEGRITY:

The **Salary** column should only contain positive numbers (no negative values or text). This is an example of domain integrity where the data type (numerical) and the range (positive integers) are enforced.

- **Valid Salary**: 50000, 60000, 55000
- **Invalid Salary**: -5000, "Fifty Thousand", "50000"

Invalid Example:

Emp_ID (PK)	First_Name	Last_Name	Department_ID (FK)	Salary
105	Moumni	Souad	2	-70000
(This violates domain integrity because the salary cannot be negative.)				

3.4 TRANSITION FROM THE E/A MODEL TO THE RELATIONAL MODEL

Step 1 (Transformation of Entities): Any entity class in the Entity/Association diagram is represented by a relation in the equivalent relational schema. The primary

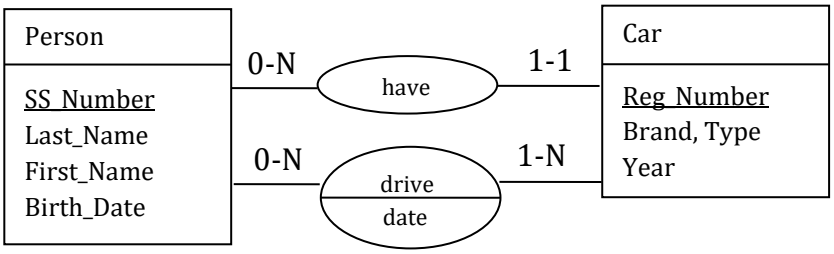
key of this relation is the identifier of the corresponding entity class. The attributes of the entity become the attributes of the relation.

Step 2 (Transformation of Associations):

- **Case 1 (1-N association)**: The primary key of the relation with a maximum cardinality of N becomes a foreign key in the other relation.
- **Case 2 (1-1 association)**: For each association where both ends have a maximum cardinality of 1, the primary key from one of the relations is included as a foreign key in the other relation.
- **Case 3 (M-N association)**: This association is transformed into a relation. The key of this new relation is composed of all the identifiers of the participating entities. The attributes of the association become the attributes of this new relation.

3.5 COMPLETE EXAMPLE

Let the following E/A model be:



The corresponding relational model is given below:

Person(SS_Number, Last_Name, First_Name, Birth_Date)

Car(Registration_Number, Brand, Type, Year, SS_Number)

Drive(SS_Number, Registration_Number, Date)

Relational Model Explanation:

A. Person (SS_Number, Last_Name, First_Name, Birth_Date):
    - This table represents person entities.

- SS_Number (Social Security Number) is the primary key in this table, which uniquely identifies each person.
- The attributes of the person are:
  - Last_Name: The surname of the person.
  - First_Name: The first name of the person.
  - Birth_Date: The birth date of the person.

B. Car (Registration_Number, Brand, Type, Year, SS_Number):
   - This table represents car entities.
   - Registration_Number is the primary key that uniquely identifies each car.
   - The attributes of the car are:
     - Brand: The brand (make) of the car (e.g., Toyota, Ford).
     - Type: The type of the car (e.g., sedan, SUV, coupe).
     - Year: The manufacturing year of the car.
     - SS_Number: This is a foreign key that refers to the SS_Number of the Person table, indicating the owner of the car.

C. Drive (SS_Number, Registration_Number, Date):
   - This table represents drives or driving events.
   - It has two foreign keys:
     - SS_Number: Refers to the Person table, identifying which person is driving.
     - Registration_Number: Refers to the Car table, identifying which car is being driven.
   - The Date attribute indicates when the driving event occurred.

- Together, SS_Number and Registration_Number form a composite primary key in this table, ensuring each driving event is uniquely identified.

D. Relationships:

1) Since we have two entities (person and car), we must have at least one relationship in the relational model. The attributes of these entities become the attributes of the new relations as follows:

Person(SS_Number, Last_Name, First_Name, Birth_Date)

Car(Registration_Number, Brand, Type, Year)

2) The association "owns" is of type (1-N), so the identifier of "person" becomes a foreign key in the car relation.

Car(Registration_Number, Brand, Type, Year, SS_Number)

3) The "drives" relation is of type (M-N), so a new relation is created. This new relation takes the same name as the association, and the "drives" attribute (date) becomes the attribute of the new relation.

Drive(SS_Number, Registration_Number, Date)

Keys:

- Primary Keys:
  - Person: SS_Number
  - Car: Registration_Number
  - Drive: Composite key (both SS_Number and Registration_Number)
- Foreign Keys:

- In Car, the SS_Number refers to the Person table, indicating the car's owner.
- In Drive, both SS_Number and Registration_Number refer to the Person and Car tables, respectively, establishing the relationship between the person driving and the car being driven.

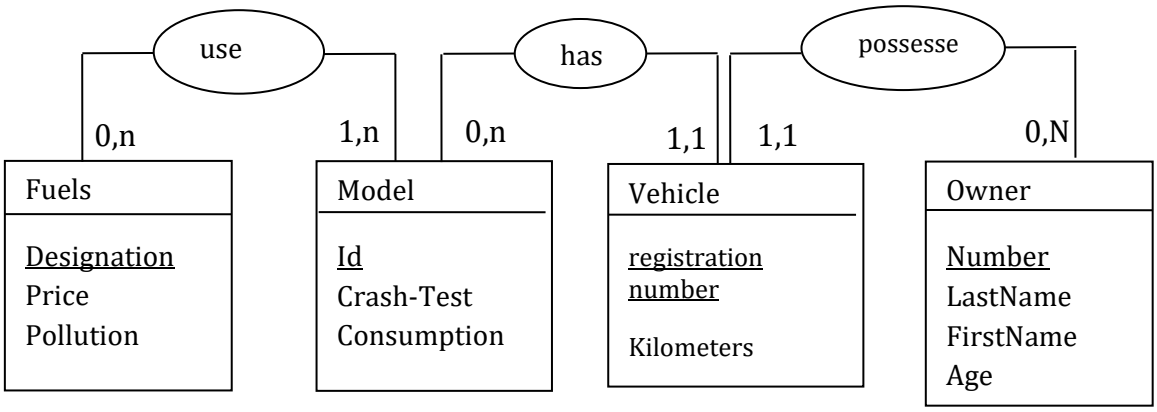
3.6 EXERCISES

3.6.1 EXERCISE 1

Provide the corresponding relational model for the exercises given in the Section 2.6.

3.6.2 EXERCISE 2

Let the following E/A model be:



Question 1: Answer with Yes/No and justify.

(a) Can a vehicle be of multiple models at the same time in this schema?

(b) Does an owner necessarily own a vehicle?

(c) Can a vehicle model use multiple fuels at the same time?

(d) A bicycle is a vehicle model that does not use fuel. Can it be represented in this model?

Question 2: Provide the corresponding relational model.

3.6.3 EXERCISE 3 (RELATIONAL MODEL FOR AN E-COMMERCE PLATFORM)

A startup is developing an online platform where customers can purchase products from different sellers. The platform needs to store and manage data about products, sellers, customers, and orders.

- Each **product** has a unique ProductID, a Name, a Description, a Price, and is sold by a **seller**.
- Each **seller** has a SellerID, Name, Email, and StoreName.
- Each **customer** has a CustomerID, Name, Email, and Address.
- A **customer can place multiple orders**, and each order can contain **multiple products**.
- An **order** has a unique OrderID, a CustomerID (who placed it), an OrderDate, and a TotalAmount.
- Each order can contain several **order items**, and each item refers to a product, with a Quantity and a PriceAtOrderTime.

Part 1: Identify Relations

1. List all the entities (relations) you would create for this system.
2. Identify **primary keys** and **foreign keys**.
3. Identify **many-to-many relationships**, if any, and show how to resolve them.

Part 2: Write the relational schema for the system.

CHAPTER 4

THE STRUCTURED QUERY LANGUAGE (SQL)

4 THE STRUCTURED QUERY LANGUAGE

SQL stands for **"Structured Query Language"**. In fact, SQL is a comprehensive language for managing relational databases. It was designed by IBM in the 1970s and has become the standard language for relational database management systems (RDBMS). It is both:

- A **data definition language** (DDL),
- A **query language** (Query language with the SELECT statement),
- A **data manipulation language** (DML; commands like UPDATE, INSERT, DELETE),
- A **data access control language** (DAC).

SQL commands often end with a semicolon (";"), which indicates the end of the command. Depending on the RDBMS used, the semicolon may or may not be mandatory.

4.1 DATA DEFINITION

When defining a column in a database table, a data type must be assigned to it, and all data stored in this column must match the type of the column. The following table summarizes the different ***basic types*** in SQL language.

Type SQL	**Meaning**
INT	Integer
FLOAT	Float
DATE	Reserves 2 digits for the month and the day, and 4 digits for the year
CHAR(size)	Character string of fixed length, where 'size' is the maximum length (in number of characters).
VARCHAR(size)	Character string of variable length between 1 and 'size'

4.1.1 CREATING A DATABASE

```
CREATE DATABASE nom_BDD
```

Example: To create the database of our higher school of economics, we execute the following command:

CREATE DATABASE **ESE_ORAN**

4.1.2 CREATING A TABLE

The CREATE TABLE statement is used to create a table by defining the name and type of each column in the table. We will limit ourselves to a simplified syntax of this statement:

```
CREATE TABLE nom_table (nom_colonne1   type1, nom_colonne2   type2, ... ) ;
```

You can add the NOT NULL option after the description of a column, which will prevent this column from containing the NULL value.

Example :

```
CREATE TABLE STUDENT(
   Registration_Num INT  NOT
NULL,
   LastName VARCHAR(10),
   FirstName VARCHAR(10),
   Date_Birth Date,
   City  VARCHAR(20),
   Code_Filière VARCHAR(10)
) ;
```

```
CREATE TABLE TRACK(
   CodeTrack VARCHAR(10)  NOT  NULL,
   Name VARCHAR(20)
) ;
```

4.1.3 CONTRAINTES D'INTEGRITE

In the definition of a table, integrity constraints can be specified for one or more columns. Every table definition must include at least one constraint of the PRIMARY KEY type.

- ✓ Primary key constraint:

```
PRIMARY KEY (column1, column2, ...) : pour une contrainte sur une table
PRIMARY KEY : pour une contrainte sur une colonne
```

Example: ***CREATE TABLE*** STUDENT(Registration_Num INT NOT NULL PRIMARY KEY, ...) or ***CREATE TABLE*** STUDENT(Registration_Num INT NOT NULL, ..., PRIMARY KEY(Registration_Num))

- ✓ Uniqueness constraint:

```
UNIQUE (column1, column2, ...): for a constraint on a table (multiple columns)
UNIQUE: for a constraint on a single column
```

- ✓ Foreign key constraint:

FOREIGN KEY (column1, column2, ...) **REFERENCES** table_ref [(col1, col2, ...)]

Example : ***CREATE TABLE*** STUDENT(Registration_Num INT NOT NULL PRIMARY KEY, ... , **FOREIGN KEY** (Code_Track) **REFERENCES** Filière(Code_Track);

Below is the set of tuples that can appear in the Student table:

STUDENT					
Registration_Num	**LastName**	**FirstName**	**Date_Birth**	**City**	**Code_Track**
200	Tabti	Fatima	25/04/2000	Oran	SE
201	Chadli	Mohammed	10/01/1999	Alger	TM
203	Chadli	Ibrahim	12/06/1999	Oran	TM

4.2 DATA MANIPULATION

The Data Manipulation Language (DML) is the language used to modify the information contained in the database.

4.2.1 INSERTION

INSERT INTO Table_Name
VALUES (Attribute_Value1, Attribute_Value2, ...) ;

Example 1: The insertion of the student 'Belhachemi Ahlem' born on 02/05/2000, residing in Oran, with the number 3707724, is done by the following SQL command:

INSERT INTO ETUDIANT
VALUES (3707724, 'Belhachemi', 'Ahlem', '02/05/2000', 'Oran');

Example 2: The insertion of the student 'Chadli' with the number 3707744 is done by:

INSERT INTO ETUDIANT(Num_Inscription,Nom) **VALUES** (3707744, Chadli);

4.2.2 MODIFICATION

The UPDATE statement allows you to modify the values of one or more fields in one or more existing rows in a table.

UPDATE Table_Name
SET attribute1 = value1, attribute2 = value2, ⋯

or

UPDATE Table_Name
SET (att1, att2,...)=(SELECT ...)
WHERE condition **;**

Example:

The student Belhachemi Ahlem now lives in Algiers. This modification is done by the following command:

UPDATE STUDENT SET City= 'Alger' WHERE Registration_Num=3707724 ;

4.2.3 DELETION

DELETE FROM Table_Name
WHERE condition ;

Example: To delete the student Belhachemi Ahlem:

DELETE FROM STUDENT WHERE Registration_Num=3707724;

4.3 DATA QUERYING

A data query language is used for searching, extracting, sorting, and formatting data from the tables in the database. The query is done using the SELECT statement, which has six different clauses, of which only the first two are mandatory. They are listed below, in the order in which they must appear when used:

SELECT [DISTINCT] column1, column2, ...
FROM table_name1, table_name2, …
[WHERE condition**]**
[GROUP BY column1, column2, ...**]**
[HAVING group_condition**]**
[ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...**]**

The table below gives the explanation of each part:

Clause	Description
SELECT	Specifies the columns to retrieve.
DISTINCT	(Optional) Eliminates duplicate rows.
FROM	Specifies the table(s) to retrieve data from.
WHERE	(Optional) Filters rows based on conditions.
GROUP BY	(Optional) Groups rows that have the same values in specified columns.

HAVING	(Optional) Filters groups (used with GROUP BY).
ORDER BY	(Optional) Sorts the result set.
LIMIT	(Optional) Restricts the number of returned rows.

In the following sections, we provide examples of how each clause is used.

4.3.1 PROJECTION

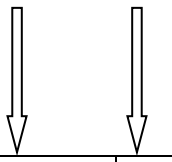
Projection consists of selecting all or certain columns from a table.

SELECT * FROM Table_Name **;** **==>** pour afficher tous les attributs d'une table

SELECT attribut1, attribut2, ... ==> pour afficher quelques attributs
FROM nom_table ;

Example: SELECT LastName,FirstName **FROM** STUDENT ;

Registration_Num	LastName	FirstName	Date_Birth	City
200	Tabti	Fatima	25/04/2000	Oran
201	Chadli	Mohammed	10/01/1999	Alger
203	Chadli	Ibrahim	12/06/1999	Oran



Vertical decomposition: *projection on the last name and first name*

LastName	FirstName
Tabti	Fatima
Chadli	Mohammed
Chadli	Ibrahim

4.3.2 RESTRICTION

The WHERE clause allows you to specify which rows to select (restriction) from a table or from the Cartesian product of multiple tables. It is followed by a predicate (a logical expression that evaluates to true or false), which will be evaluated for each row. Rows for which the predicate is true will be selected.

SELECT attribute1, attribute2, ...
FROM Table_Name
WHERE predicate **;**

We will limit ourselves to comparison operators: =, <>, <, >, <=, >=, and to logical operators: AND, OR et NOT.

Example : Display the students who do not live in Oran

SELECT * **FROM** STUDENT **WHERE NOT(**CITY="ORAN") **;**

Registration_Num	LastName	FirstName	Date_Birth	City
200	Tabti	Fatima	25/04/2000	Oran
201	Chadli	Mohammed	10/01/1999	Alger
203	Chadli	Ibrahim	12/06/1999	Oran

⇨

Registration_Num	LastName	FirstName	Date_Birth	City
201	Chadli	Mohammed	10/01/1999	Alger

Horizontal decomposition based on a condition

4.3.3 SORTING

The rows constituting the result of a SELECT query are returned in an indeterminate order. The ORDER BY clause specifies the order in which the selected rows will be given. By default, the results are sorted in ascending order (ASC), but it is possible to reverse the order by using the DESC suffix after the column name.

Example: Display the list of students ordered by ascending last name and descending first name.

SELECT * **FROM** STUDENT **ORDER BY** LastName, FirstName DESC **;**

4.3.4 JOIN

When multiple tables are specified in the FROM clause, the Cartesian product of the tables is obtained. This Cartesian product is generally of little interest; it is the WHERE clause that restricts the result to retain only the relevant rows.

Example: Display the names of the students along with the names of their majors.

SELECT LastName, Name **FROM** STUDENT, TRACK

WHERE STUDENT.codeTrack=TRACK. codeTrack;

Example:

Table STUDENT					
Reg Num	**LastName**	**FirstName**	**Date_Birth**	**City**	**CodeTrack**
200	Tabti	Fatima	25/04/2000	Oran	MI
201	Chadli	Mohammed	10/01/1999	Alger	MI
203	Chadli	Ibrahim	12/06/1999	Oran	ECO

Table TRACK	
codeTrack	**Name**
MI	Math-INFO
ECO	Economie

Result:

LastName	Name
Tabti	Math-INFO
Chadli	Math-INFO
Chadli	Economie

Special case (Joining a table with itself): It allows combining information from one row of a table with information from another row of the same table. In this case, at least one instance of the table must be renamed with an alias to unambiguously prefix each column name.

Example: Display the first and last names of the students who live in the same city as "Tabti".

SELECT LastName, FirstName **FROM** STUDENT E1, STUDENT E2

WHERE E1.City = E2. City **AND** E1.LastName="Tabti" **AND** E2. LastName <> "Tabti" ;

E1				
Reg Num	**LastName**	**FirstName**	**Date_Birth**	**City**
200	*Tabti*	*Fatima*	*25/04/2000*	*Oran*
201	Chadli	Mohammed	10/01/1999	Alger
203	Chadli	Ibrahim	12/06/1999	Oran

E2				
Reg Num	**LastName**	**FirstName**	**Date_Birth**	**City**
200	*Tabti*	*Fatima*	*25/04/2000*	*Oran*
201	Chadli	Mohammed	10/01/1999	Alger
203	*Chadli*	*Ibrahim*	*12/06/1999*	*Oran*

Result:

LastName	FirstName
Chadli	*Ibrahim*

4.3.5 CALCULATION FUNCTIONS

In SQL, **calculation functions** refer to functions that perform mathematical or aggregate operations on data. These can be used to manipulate numeric data, perform mathematical operations, or summarize information. Here are some common types of calculation functions in SQL:

Mathematical Functions:

- **ABS()**: Returns the absolute value of a number.
- **POWER()**: Raises a number to a specified power.
- **CEILING()**: Returns the smallest integer greater than or equal to the number.
- **FLOOR()**: Returns the largest integer less than or equal to the number.

- **ROUND()**: Rounds a number to a specified number of decimal places.

Aggregate Functions:

- **SUM():** Calculates the total sum of a numeric column.
- **AVG():** Calculates the average value of a numeric column**.**
- **COUNT():** Returns the number of rows (or non-null values) in a column.
- **MIN():** Returns the smallest value in a numeric column.
- **MAX():** Returns the largest value in a numeric column.

String Functions (for string manipulation):

- **LENGTH():** Returns the length of a string**.**
- **CONCAT():** Concatenates two or more strings together.

Date Functions (used for date calculations):

- **DATEDIFF():** Returns the difference between two dates.
- **DATEADD():** Adds a specific time interval to a date**.**

Example 1: Display the number of students

SELECT count(*) as NbStudents **FROM** STUDENT ;

Result:

NbStudents
3

Example 2: Display the maximum registration number

SELECT max(Reg_Number) as maxinum **FROM** STUDENT ;

Result:

Maxinum
203

4.3.6 THE GROUP BY CLAUSE

The GROUP BY statement groups rows that have the same values into summary rows.

Example: Group students by city

```
SELECT City, COUNT(*) AS NB_ STUDENT
FROM STUDENT
GROUP BY City ;
```

Result:

City	NB_STUDENT
Oran	2
Alger	1

4.3.7 THE HAVING CLAUSE

Example: Group students by **city** and only show the cities that have **more than one student**.

```
SELECT City, COUNT(*) AS NB_STUDENT
FROM STUDENT
GROUP BY City
HAVING COUNT(*) > 1;
```

Explanation:

- GROUP BY Ville → groups the students by their city.
- COUNT(*) → counts how many students are in each group (i.e. each city).
- HAVING COUNT(*) > 1 → filters out groups (cities) that have **1 or 0** students. We only keep those with **more than 1**.

Result:

City	NB_STUDENT
Oran	**2**

Because Oran has two students (Fatima and Ibrahim), it appears in the result.

4.4 EXERCISES

4.4.1 EXERCISE 1

The management of a private school has decided to computerize the management of timetables. Each student is characterized by their student number, last name, first name, and age. Each course is uniquely identified by a code (INFO2, MATH1, ...) and has a title (microeconomics, algebra, ...) as well as a responsible teacher. The number of sessions for each course is also known. Teachers are characterized by an alphanumeric code, their last name, first name, and rank. Finally, each session is identified by the course and

the session number (session 3 of the INFO2 course, session 1 of the MATH1 course, ...), the date, the start time, and the end time during which the session takes place, as well as the room and the teacher who conducts the session. Students enroll in the courses they wish to attend.

Adopted relational schema:

student (id, lastName, firstName, age)

teacher (teacherId, lastName, firstName, rank)

course (courseCode, title, teacherId, numberOfSessions)

session (courseCode, sessionNumber, date, room, startTime, endTime, teacherId)

enrollment (studentId, courseCode**)**

Question: Provide the SQL queries to answer the following questions:

1. Create the tables "Student" and "Session".
2. Add the teacher ('INFO2', 'Houcine', 'Mohammed').
3. Enroll the student ('l0372', 'Cherifi', 'Ayman', 20) in the course ('ALG2', 'Algebra', 'EN023').
4. Display the list of students.
5. Find the name and surname of all students under 20 years old.
6. Find the name and surname of the teacher responsible for the Statistics course.
7. Find the name and surname of all students enrolled in the Probability or Statistics course.
8. Who are the youngest students in the class?
9. Determine the number of teachers involved in the Probability course.
10. Where and when is the first Accounting class held?

11. Display a "timetable" for the Statistics course.
12. The grade of the teacher "Senouci Zaki" has changed to "Maître-de-conférence B".
13. Add a lecture for Accounting on December 14th with Senouci Zaki in room S250 from 14:00 to 18:00.
14. The student ('l0372', 'Cherifi', 'Ayman') is no longer enrolled in the 'ALG2' course.

4.4.2 EXERCISE 2

Let us take the relational model of the exercise 3 (Relationnele chapitre)

Give the SQL queries to answer the following questions:

1. List the names and emails of all customers who have placed at least one order.
2. Retrieve the product names and quantities in the order with OrderID = 102.
3. List all products that have **never been ordered**.
4. Find the total sales (sum of quantities × price at order time) for each product.
5. For each seller, show the total revenue generated from their products.

4.4.3 EXERCISE 3

Let us take the following relational model:

Fuel(Designation, Price, Pollution)
Model(Id, CrashTest, Consumption)
Vehicle(RegistrationNumber, Kilometers, ModelId, OwnerNumber)
Owner(Number, LastName, FirstName, Age)
Uses(Designation, ModelId)

A) Basic Queries

1. List all fuel types along with their price and pollution level.
2. Show all vehicle registration numbers and the number of kilometers they've traveled.

B) Join Queries

3. List the models along with the fuel they use.
4. List all vehicles with their owner's name and the fuel type they use.

C) Conditions and Filtering

5. Find all models that use fuel types with pollution greater than 120.
6. Find all owners under 30 years old who own at least one vehicle.

D) Aggregation

6. Count how many vehicles use each fuel type.
8. Find the average consumption of models using 'Diesel'.

D) Subqueries

9. List owners who own a vehicle that uses the least polluting fuel.
10. Find vehicles whose models failed the crash test and use fuel costing more than 46 DA.

CHAPTER 5

SQL QUERIES WITH ACCESS

5 SQL QUERIES WITH ACCESS

5.1 INTRODUCTION

Microsoft Access is a powerful database management system (DBMS) that combines the relational database engine with a graphical user interface (GUI) and software-development tools. It is a part of the Microsoft Office suite and is used for storing, managing, and analyzing data in a structured manner. Access is particularly suitable for small to medium-sized databases and is widely used by businesses, educational institutions, and other organizations for creating and managing their databases. Access can easily integrate with other Microsoft Office applications like Excel, Outlook, and Word, enabling users to import/export data, generate reports, and automate tasks seamlessly.

5.2 KEY FEATURES OF MICROSOFT ACCESS

1. Relational Database Management:
    - Access allows users to create relational databases, which store data in separate tables that are linked by relationships. This makes it easier to organize, query, and update data.
2. Tables:
    - Tables in Access are the basic units of data storage. They contain rows (records) and columns (fields) where data is stored. A table might represent a single entity (e.g., Students, Courses, Enrollments), and it is structured to minimize redundancy and optimize queries.
3. Queries:
    - Queries are used to retrieve specific data from one or more tables. You can use queries to filter, sort, and manipulate the data. Access supports SQL queries (Structured Query Language), which allow for complex querying and data manipulation.
    - Access also provides a Query Design View, allowing users to build queries visually.

4. Forms:
    - Forms in Access are used for data entry, making it easier to add, edit, or view data in a user-friendly interface. Forms are highly customizable and can include buttons, dropdown menus, and other controls to simplify data entry.
5. Reports:
    - Reports allow users to print data in a structured, readable format. Reports are useful for summarizing data, generating invoices, or producing any other form of structured output based on database information.
6. Macros and VBA (Visual Basic for Applications):
    - Access allows users to automate tasks using macros or VBA programming. This makes it possible to create custom operations, automate report generation, or even perform batch updates.
7. Data Types:
    - Access supports various data types such as text, numbers, dates, and binary data (e.g., images, files). This flexibility allows users to store different kinds of information efficiently.
8. Relationships:
    - Relationships define how tables are linked together. Access supports one-to-many, many-to-many, and one-to-one relationships, allowing users to maintain referential integrity and avoid data duplication.
9. Security:
    - Access provides tools to secure databases through password protection, user-level security (with customized permissions for different users), and encryption.

5.3 Typical Use Cases of Microsoft Access:

- Business Applications: Many small businesses use Access to manage customer data, inventory, orders, and billing.
- Academic Databases: Educational institutions use Access to store student records, faculty information, and course registrations.
- Non-Profit Organizations: Non-profits often use Access to manage donor data, event registrations, and volunteer schedules.
- Personal Projects: Individuals can use Access to track personal finances, library inventories, or even collections like books or movies.

5.4 Case Study: Student Enrollment System Using Microsoft Access

5.4.1 Overview

A university wants to manage student enrollments using a database system built in Microsoft Access. The system will store information about **students**, **courses**, **enrollments**, and **instructors**. Students can enroll in multiple courses, and each course can have multiple students. Each course is taught by a single instructor.

The university administration needs this system to:

- Keep track of student personal data.
- Manage courses and assign instructors.
- Record which students are enrolled in which courses.
- Store final grades for enrolled students.

Objectives

1. Identify and define the entities and relationships.
2. Create an E/R diagram.
3. Implement the E/R model using Microsoft Access tables.
4. Create queries to extract meaningful information.

5.4.2 DESIGN PHASE

Step 1: Build the entity/relationship model

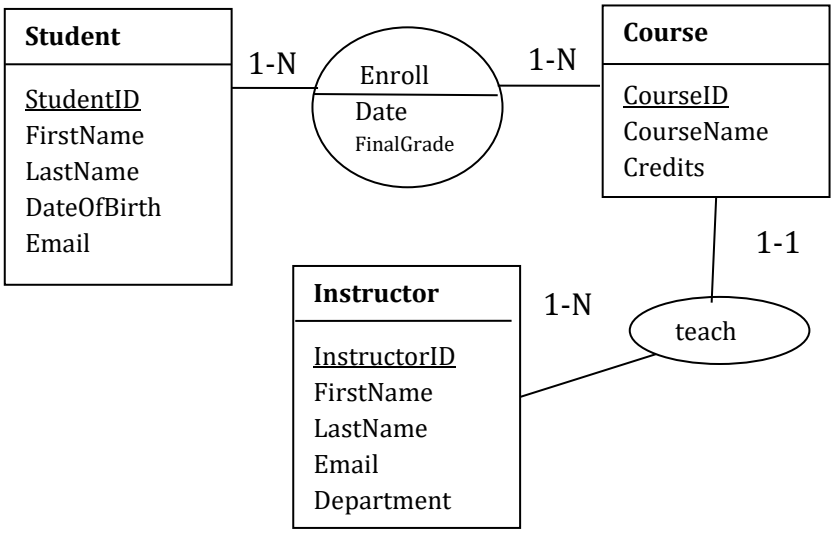
Entities and Attributes

1. **Student**
    - StudentID (Primary Key)
    - FirstName
    - LastName
    - DateOfBirth
    - Email
2. **Instructor**
    - InstructorID (Primary Key)
    - FirstName
    - LastName
    - Email
    - Department
3. **Course**
    - CourseID (Primary Key)
    - CourseName
    - Credits
4. **Enrollment**
    - EnrollmentID (Primary Key)
    - StudentID (Foreign Key)
    - CourseID (Foreign Key)
    - EnrollmentDate
    - FinalGrade

Relationships

- A **student** can enroll in many **courses**: M:N (Resolved with the **Enrollment** table).
- A **course** can have many **students**: M:N.

- A **course** is taught by one **instructor**, but an instructor can teach multiple **courses**: 1:N



Step 2: Convert the entity/relationship model into a relational model

- **Student**(StudentID, FirstName, LastName, DateOfBirth, Email)
- **Instructor**(InstructorID, FirstName, LastName, Email, Department)
- **Course**(CourseID, CourseName, Credits, InstructorID)
- **Enrollment**(StudentID, CourseID, EnrollmentDate, FinalGrade)

5.4.3 Implementation Phase with Access

Step 1: Creating a new database

1. Open Microsoft ACCESS

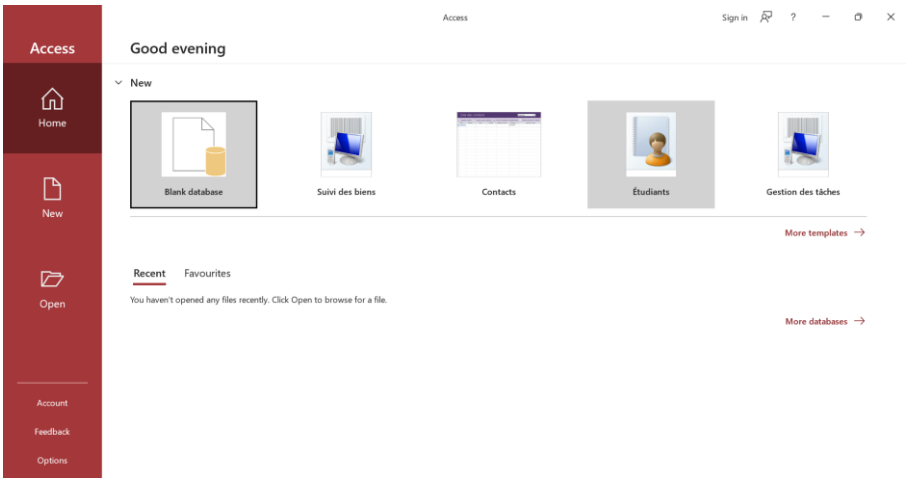


Figure 4 Opening Access

2. Click on blank database, and then chose the name of the database “student.accdb” and the directory in which the file will be saved.

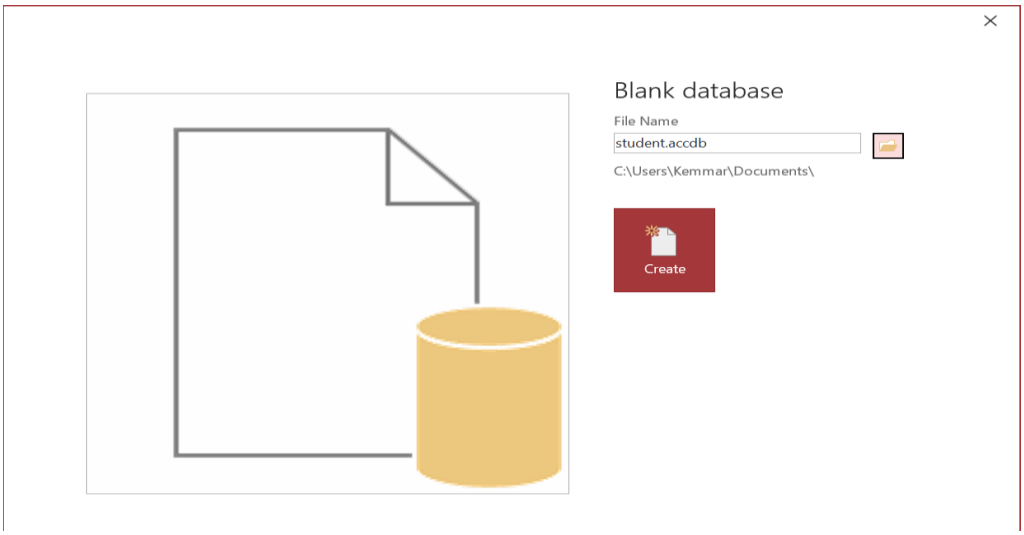


Figure 5 Creating a new database

Step 2: Creating tables

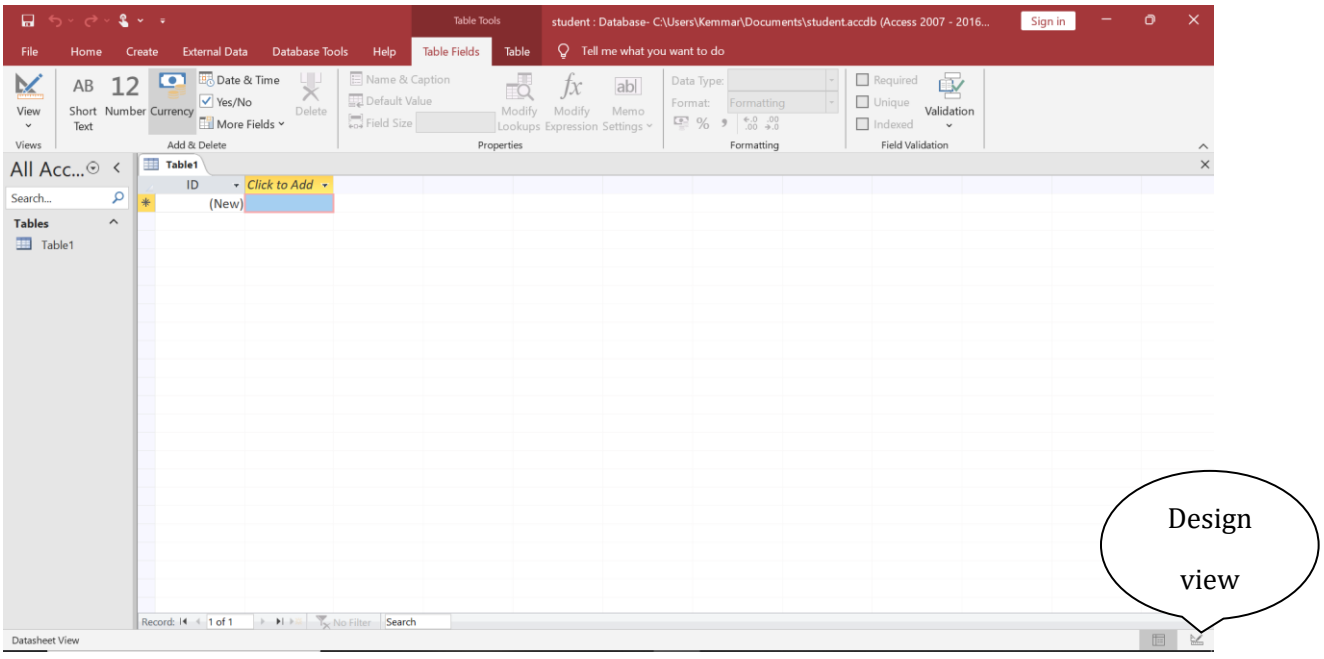


Figure 6 Creating a new table

- You have to save the current table by clicking on (ctl + S) and then choosing the name “Student”

- Switch to Design View and select the corresponding type for each attribute as follows:

Note: each table should have at least one attribute designed as a primary key

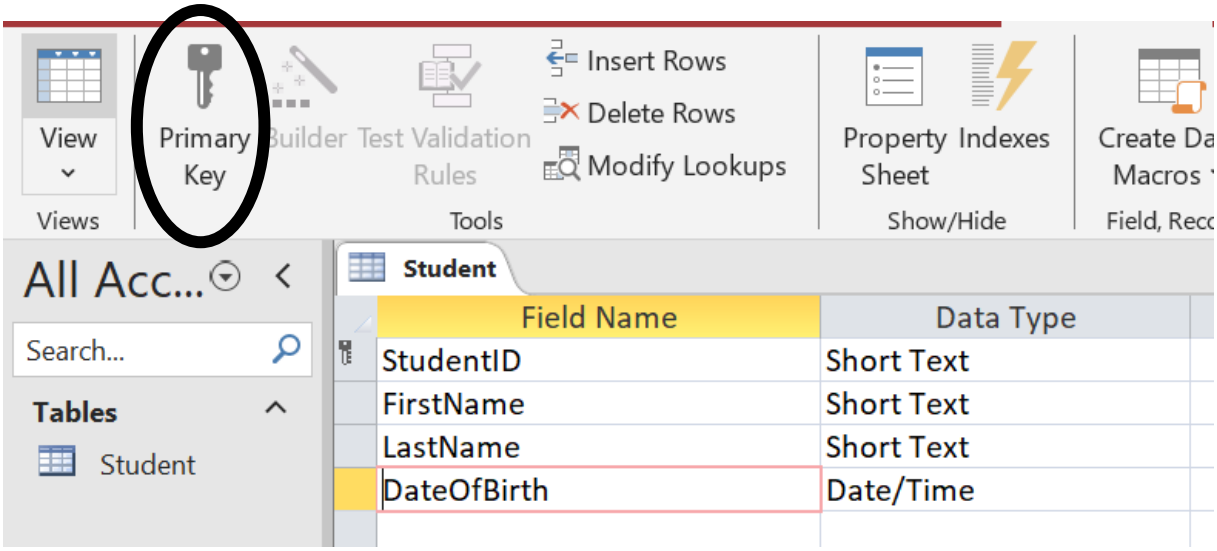


Figure 7 insertion of fields

To create the second table, select Menu ➔Create➔Table

You have to save the new table (ctrl+S) with the name Course

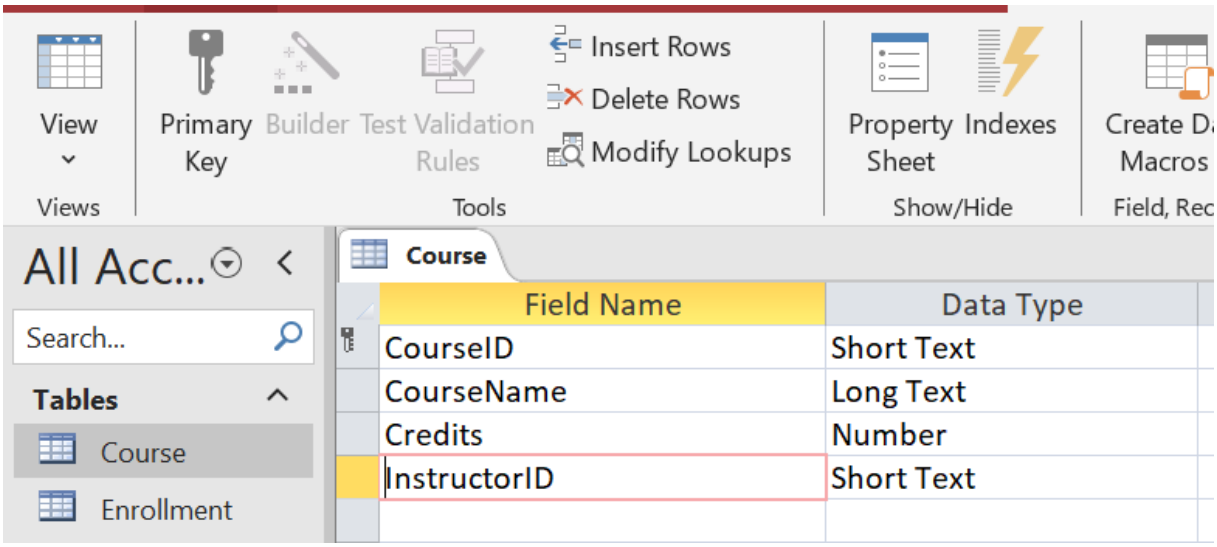


Figure 8 Creating the table Course

Create the tables Instructor and Enrollment as follows (by following the same above steps):

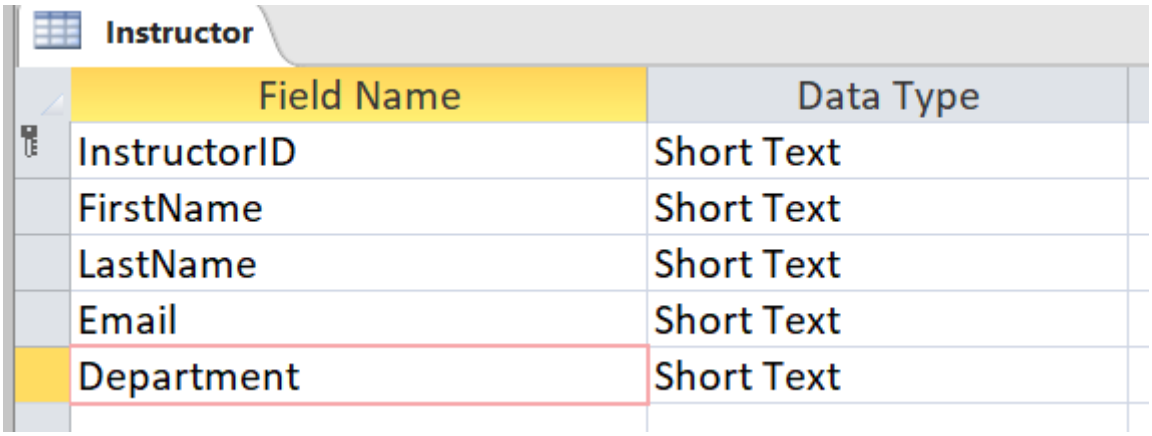


Figure 9 Creating the table Instructor

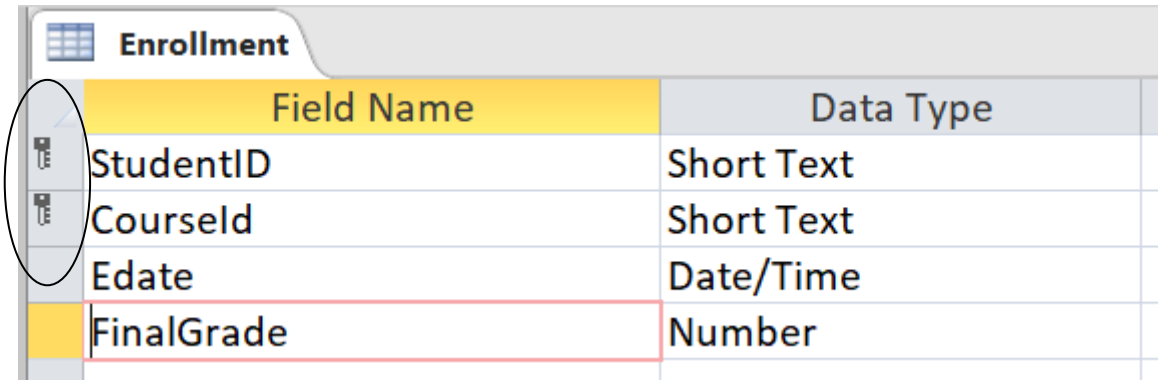


Figure 10 Creating the table enrollment

Step 3: Create, Edit or Delete a relationship

A relationship in Access helps you combine data from two different tables. Each relationship consists of fields in two tables with corresponding data. When you use related tables in a query, the relationship lets Access determine which records from each table to combine in the result set. A relationship can also help prevent missing data, by keeping deleted data from getting out of synch, and this is called referential integrity.

When you create a relationship between tables, the common fields are not required to have the same names, although it is often the case that they do. The common fields must have the same data type. If the primary key field is an AutoNumber field, however, the foreign key field can also be a Number field if the **FieldSize** property of both fields is the same. For example, you can match an AutoNumber field and a Number field if the **FieldSize** property of both fields is Long Integer. When both common fields are Number fields, they must have the same **FieldSize** property setting.

Create a table relationship by using the Relationships window

1. On the **Database Tools** tab, in the **Relationships** group, click **Relationships**.
2. On the **Relationships Design** tab, in the **Relationships** group, click **Add Tables**.
3. Select one or more tables or queries and then click **Add**. After you have finished adding tables and queries to the Relationships document tab, click **Close**.
4. Drag a field (typically the primary key) from one table to the common field (the foreign key) in the other table. To drag multiple fields, press the CTRL key, click each field, and then drag them.

The **Edit Relationships** dialog box appears.

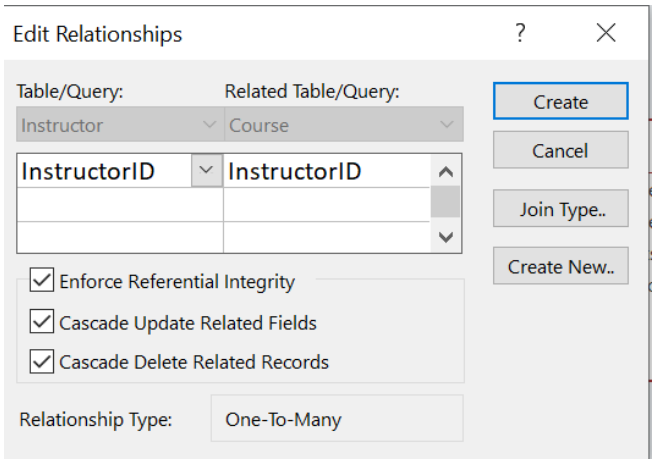


Figure 11 How to edit relationships

5. Verify that the field names shown are the common fields for the relationship. If a field name is incorrect, click on the field name and select the appropriate field from the list.

   To enforce referential integrity for this relationship, select the **Enforce Referential Integrity** check box.
6. Click **Create**.

Access draws a relationship line between the two tables. If you selected the **Enforce Referential Integrity** check box, the line appears thicker at each end. In addition, again only if you selected the **Enforce Referential Integrity** check box, the number **1** appears

over the thick portion on one side of the relationship line, and the infinity symbol (∞) appears over the thick portion on the on the other side of the line, as shown in the following figure.

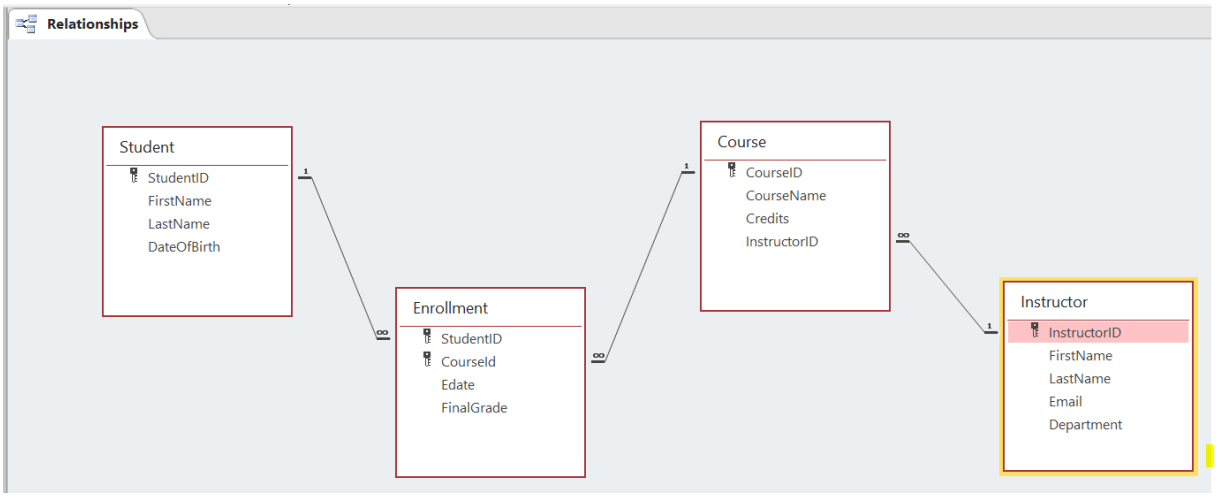


Figure 12 Insert relations between tables

Step 4: Data Insertion

Method 1: Using Datasheet View (Graphical Interface)

- Open your Access database.
- In the Navigation Pane, double-click the table where you want to insert data.
- The table opens in Datasheet View.
- Scroll to the bottom of the table and enter the data into the new (blank) row.
- Press Enter or click away to save the new record.

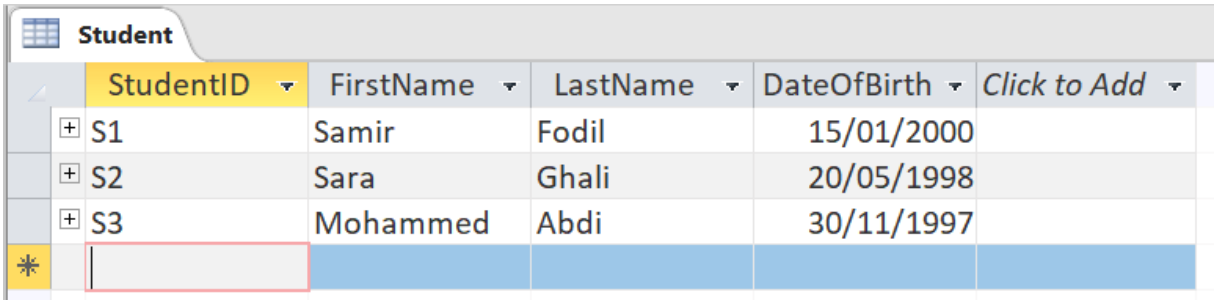


Figure 13 How to insert a new student

Method 2: Using SQL (Query)

Steps to Run SQL in Access:

1. Go to the **"Create"** tab in the ribbon.
2. Click **"Query Design"**.
3. Close the "Show Table" dialog (unless needed).
4. Switch to **SQL View** (click on the **"SQL View"** button).
5. Type or paste your INSERT INTO query.
6. Click **"Run"** (red exclamation point button).

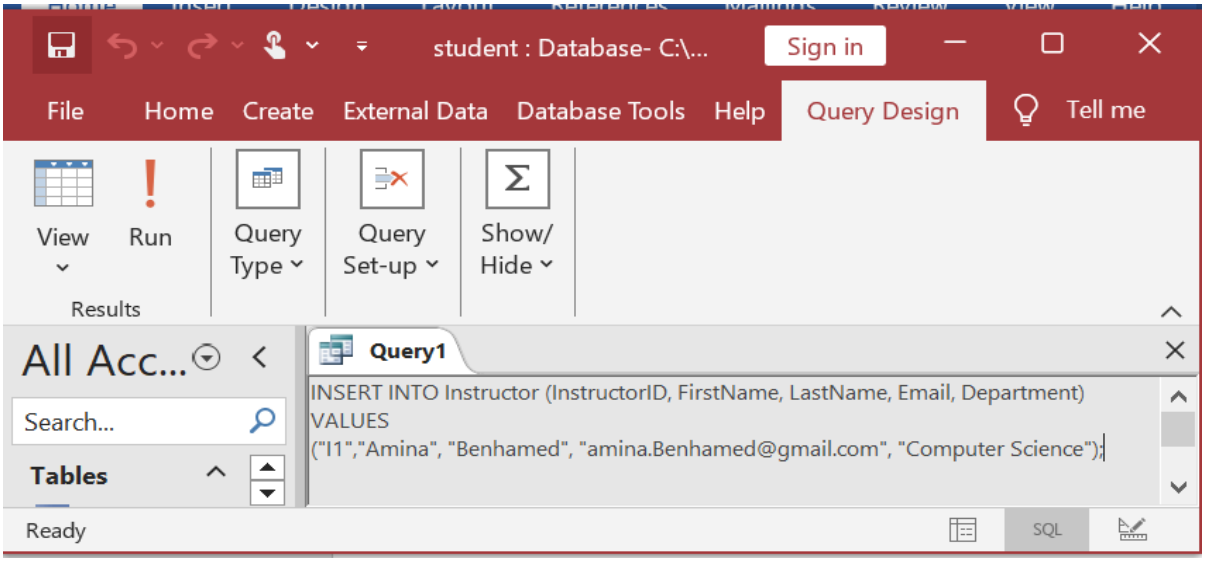


Figure 14 Creating a new SQL query

Insert students

```
INSERT INTO Students (StudentID, FirstName, LastName, DateOfBirth)
VALUES
('S1','Samir', 'Fodil', '2000-01-15'),
('S2','Sara', 'Ghali', '1998-05-20'),
('S3','Mohammed', 'Abdi', '1997-11-30');
```

Insert Instructors Date

```
INSERT INTO Instructor (InstructorID, FirstName, LastName, Email, Department)
VALUES
('I1','Amina', 'Benhamed', 'amina.Benhamed@gmail.com', 'Computer Science'),
('I2', 'Omar', 'Mehdi', 'Economy', 'Omar.Mehdi@gmail.com'),
('I3', 'Asmaa', 'Rachdi', 'Computer Science', 'Asmaa.rachdi@gmail.com');
```

Insert Courses Data

```
INSERT INTO Courses (CourseID, CourseName, Credits, InstructorID)
VALUES
('C1','Computer Science', 3, 'I1'),
('C2', 'Data Structures', 4, 'I2'),
('C3', 'Database Management Systems', 3, 'I3');
```

Insert Enrollments Data

```
INSERT INTO Enrollments (StudentID, CourseID, Edate, FinalGrade)
VALUES
(S1, C1, '2025-01-01', 12 ),
(S1, C2, '2025-01-01', 14),
(S2, C3, '2025-02-01', 16),
(S3, C1, '2025-03-01', 9);
```

5.5 USING QUERIES

In Microsoft Access, **queries** are powerful tools used to retrieve, filter, and manipulate data stored in one or more tables. They act like questions posed to the database, allowing users to extract meaningful information from large datasets. Access provides several ways to create queries, each tailored to different user needs and levels of expertise. Understanding these methods is essential for effective data analysis and reporting.

1. Query Design View

This is the most commonly used method. It offers a graphical interface where users can:

- Select tables and fields visually.
- Define sorting and filtering criteria.

- Use expressions for calculated fields.
- Join tables to combine related data.

Ideal for users who prefer a visual and structured way to build queries without writing SQL code.

2. SQL View

For more advanced users, SQL View provides direct access to the SQL (Structured Query Language) behind a query. This method allows:

- Writing custom SQL statements.
- Creating more complex queries not possible in Design View.
- Greater flexibility and control.

It is particularly useful for those familiar with database concepts and SQL syntax.

3. Query Wizard

The Query Wizard guides users through a step-by-step process to create simple queries such as:

- Select queries
- Crosstab queries
- Find duplicates
- Find unmatched records

This method is beginner-friendly and helps users create queries quickly with minimal technical knowledge.

4. Using Macros or VBA (Advanced)

For automation or advanced logic, queries can also be created and executed through:

- **Macros**, which automate repetitive tasks.

- **Visual Basic for Applications (VBA)** code, allowing dynamic query generation and execution.

These methods are powerful but require programming knowledge and are generally used in complex database applications.

Each of these methods has its own strengths, and choosing the right one depends on the complexity of the task and the user's familiarity with Access and SQL. Together, they make Access a versatile tool for both novice and advanced database users.

In the following, we give the different steps allowing te create queries using the two first methods.

5.5.1 Queries with the Query Design View

Creating a basic Select Query

1. With a database open, go to the “Create” tab at the top of the Access window.
2. Select the “Query Design” button from the ribbon below.

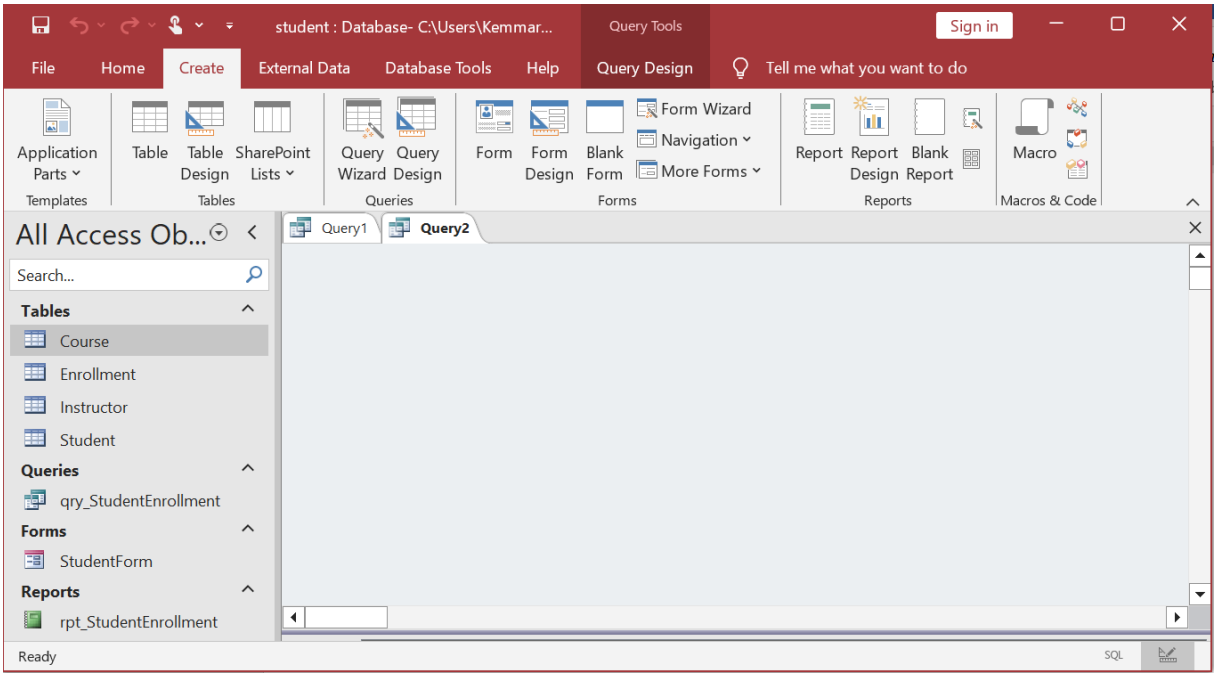


Figure 15 Creating a new query with the Query Design View

3. You will see a screen like the example below. Select the tables you want to use in your query by double-clicking on their names or highlighting them and clicking "Add." You can select more than one table but they need to be related.

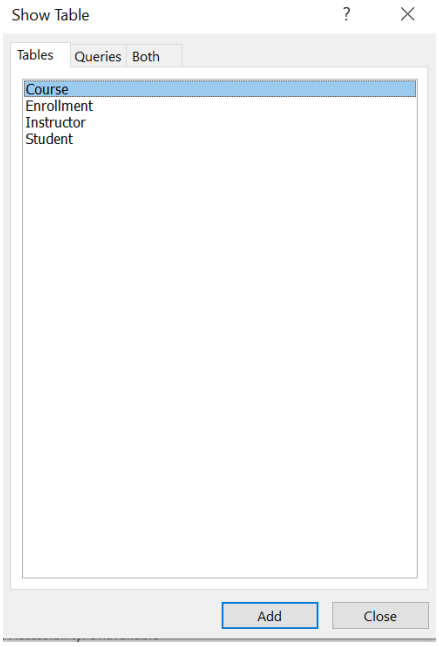


Figure 16 Choosing tables

4. For example, let us select the student table, the selected table will appear as a small window in the **Object Relationship pane**. In the table window, double-click the **field names** you want to include in your query. They will be added to the **design grid** in the bottom part of the screen.

5. Set the **search criteria** by clicking the cell in the **Criteria:** row of each field you want to filter. Typing criteria into more than one field in the Criteria: row will set your query to include only results that meet all criteria.

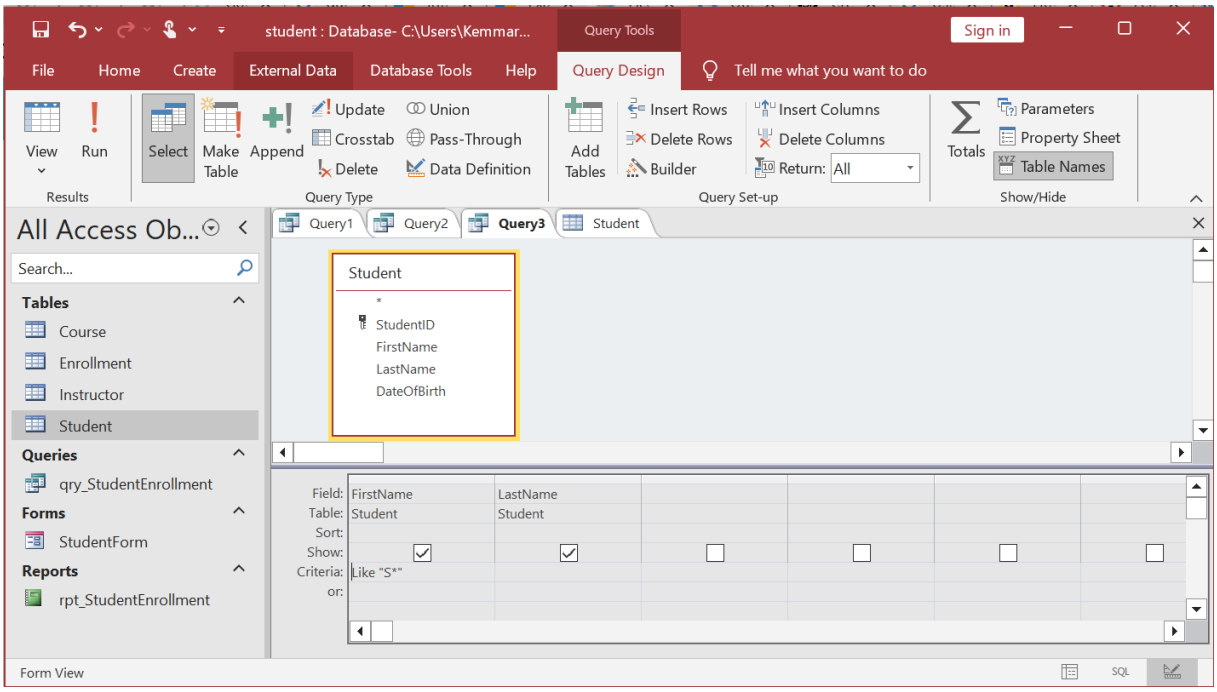


Figure 17 How to choose criteria

6. After you have set your criteria, **run** the query by clicking the **Run** command on the **Design** tab.

7. The query results will be displayed in the query's **Datasheet view**, which looks like a table. If you want, **save** your query by clicking the **Save** command in the Quick Access Toolbar. When prompted to name it, type the desired name, then click **OK**.

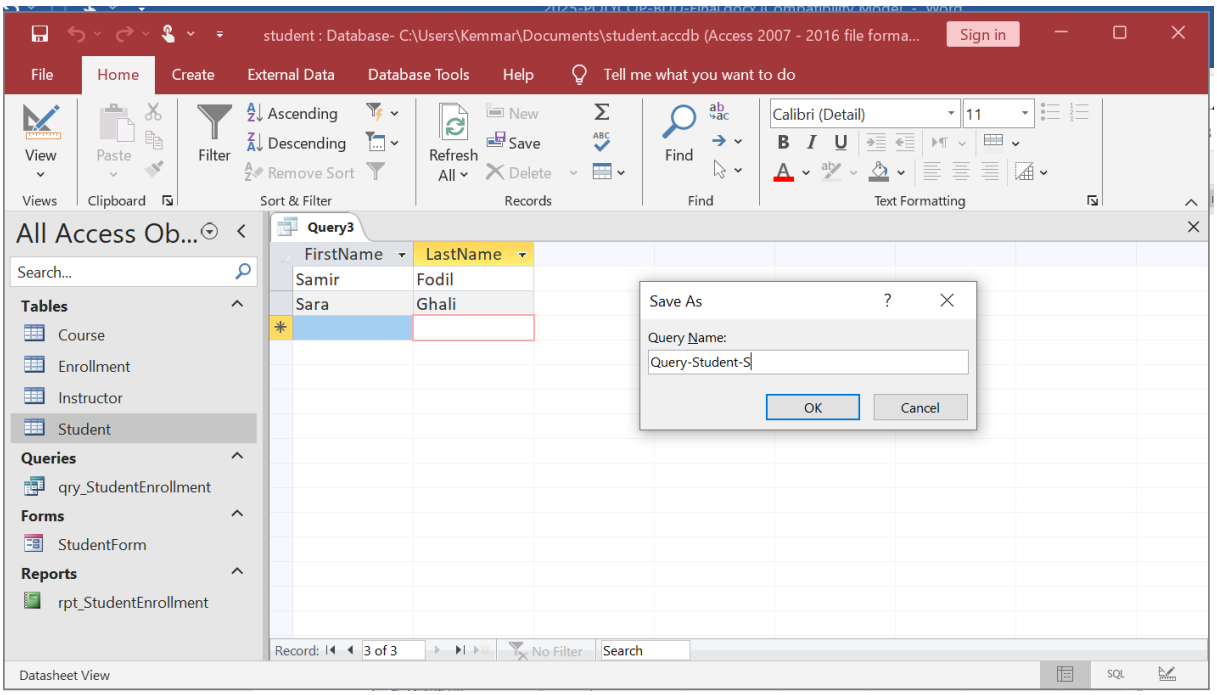


Figure 18 How to name a query

5.5.2 *QUERIES WITH SQL*

To switch to **SQL mode in Microsoft Access**, follow these steps:

1. Open your Access database.
2. Go to the **"Create"** tab on the Ribbon.
3. Click on **"Query Design"**.
4. If the **"Show Table"** dialog appears, you can either:
    - Add the relevant table(s), or
    - Close the dialog if you want to write SQL manually.
5. On the top-left, click the **"View"** button (in the toolbar).
6. From the dropdown, select **"SQL View"**.

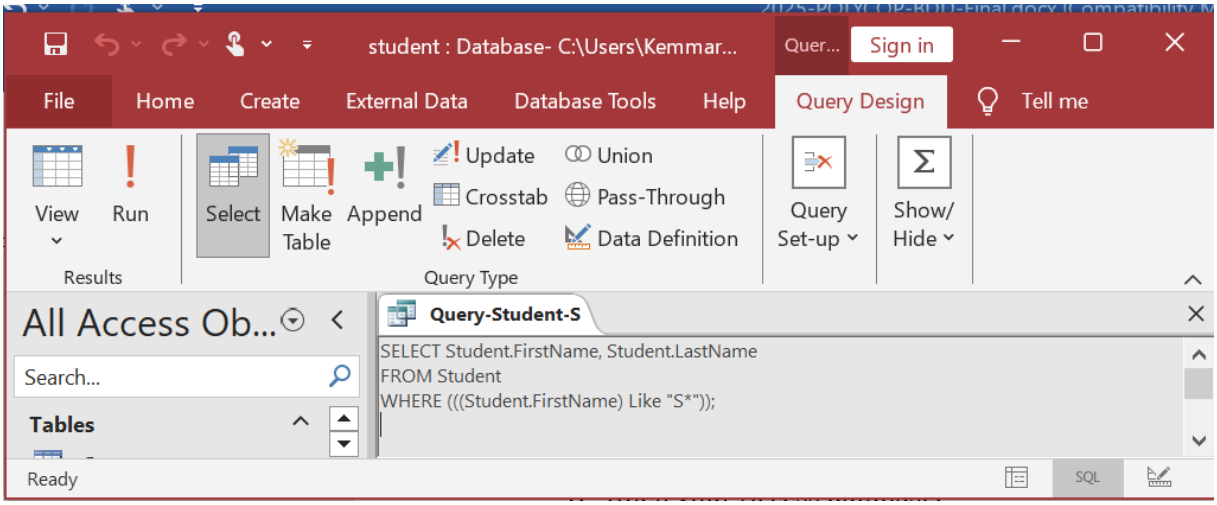


Figure 19 Add a new SQL query with SQL view

Write the following queries in Microsoft Access:

1. List all courses.

```
SELECT  Course.*
FROM    Course;
```

2. List the first name and last name of all instructors.

```
SELECT   Instructor.FirstName, Instructor.LastName,
FROM     Instructor;
```

3. List all students enrolled in a course called "Computer Science".

```
SELECT   Students.FirstName, Students.LastName, Courses.CourseName
FROM     Students, Enrollments
WHERE Students.StudentID = Enrollments.StudentID
AND  Enrollments.CourseID = Courses.CourseID
         Courses.CourseName = 'Computer Science';
```

4. Show all courses with their instructors' names.

```
SELECT  Course.*, Instructor.LastName
FROM     Course, Instructor
WHERE Course.InstructorID=Instructor.InstructorID;
```

5. Get a list of courses a specific student (e.g., Sara Ghali') is enrolled in:

```
SELECT   Courses.CourseName
FROM  Students, Enrollments
WHERE Students.StudentID = Enrollments.StudentID
        AND Enrollments.CourseID = Courses.CourseID
        AND Students.FirstName = 'Sara' AND Students.LastName = 'Ghali';
```

6. Get the total number of students enrolled in each course:

```
SELECT Courses.CourseName, COUNT(Enrollments.StudentID) AS
TotalStudents
FROM Courses, Enrollments
WHERE  Courses.CourseID = Enrollments.CourseID)
GROUP BY  Courses.CourseName;
```

7. Update the enrollment date for a student (e.g., Sara Ghali in 'Data Structures'):

```
UPDATE Enrollments
SET EnrollmentDate = '2025-01-10'
WHERE StudentID = 'S2' AND CourseID ='C2';
```

8. Delete an enrollment record for a student (e.g., Sara Ghali in 'Data Structures'):

```
DELETE FROM Enrollments
WHERE StudentID = 'S2' AND CourseID = 'C2';
```

5.6 Creating Forms for Data Entry

In Microsoft Access, forms are objects that provide a user-friendly interface for entering, modifying, viewing, and navigating data stored in tables or queries. They serve as customized windows through which users interact with the database, without having to deal directly with the raw tables.

How to Create Forms in Access:

1. Open Access and go to the **Create** tab.
2. Click on **Form Design** to create a new blank form.

3. Use the **Form Wizard** or **Design View** to add fields for students, courses, and enrollments.
4. Customize the form with buttons for navigating, adding, or updating records.

Demonstration: Create a *student form* to enter new student records, including fields for FirstName, LastName, and DOB.

A) Select the student table and then select insert a form, you should obtain the following result:

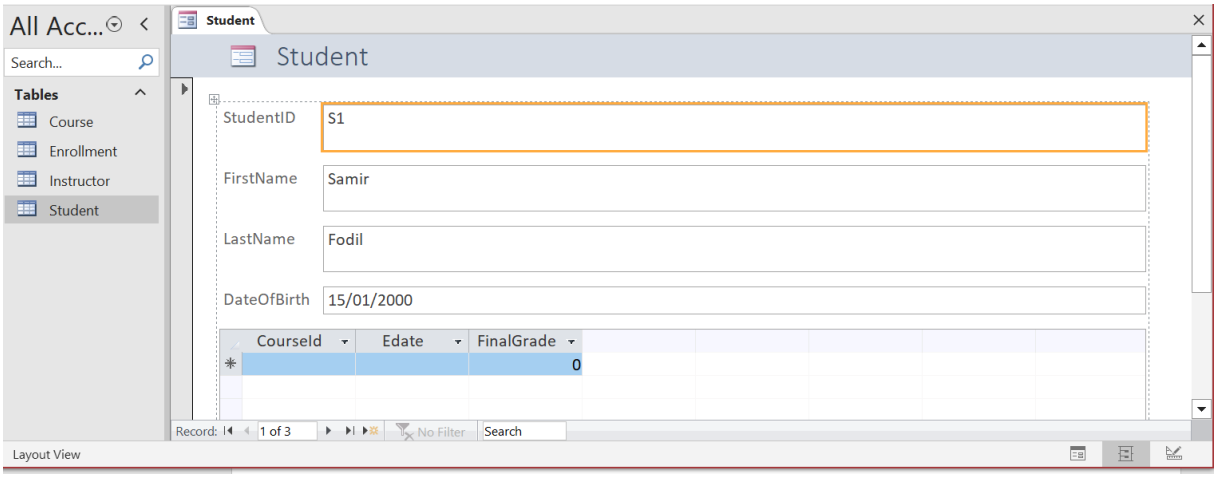


Figure 20 Creating a new form

B) Now, you can do some modifications, for example change the name of attributes and delete the grid located at the bottom.

C) Save the form (Ctrl+S) with the name StudentForm

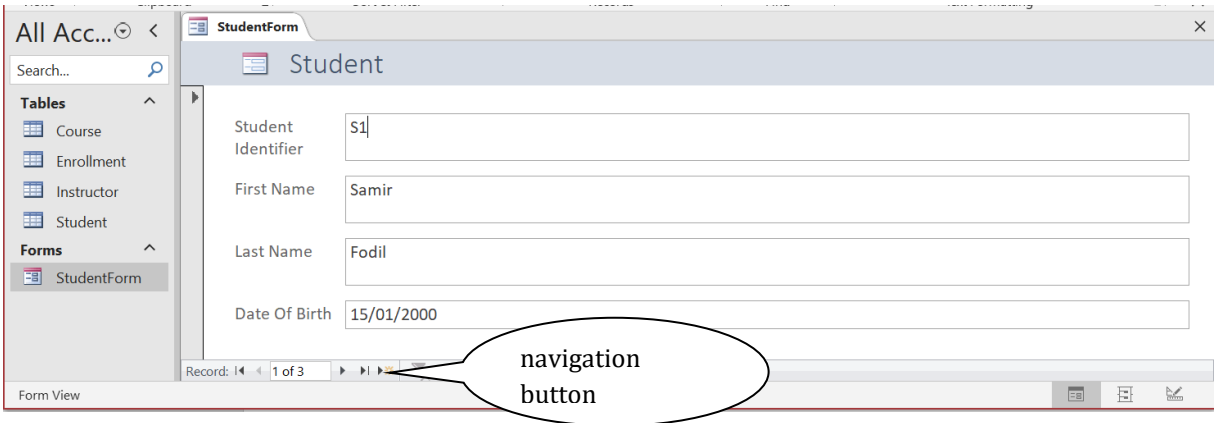


Figure 21 Modifying some fields of the form

5.7 REPORTING

In **Microsoft Access**, **reporting** refers to the process of **organizing, summarizing, and presenting data** in a structured and printable format. A **report** is a database object that is typically used to display data from tables or queries in a way that's optimized for **printing, sharing, or analysis**.

What Reports in Access Are Used For:

- Creating **invoices**, **summaries**, **sales reports**, **customer lists**, etc.
- Displaying data with **grouping**, **sorting**, **totals**, and **calculations**.
- Generating **print-ready** documents or **PDFs**.
- Presenting **visual summaries** with **charts**, **labels**, and **headers/footers**.

Demontration 1: Student Enrollment Report

Goal: Show a list of students with the courses they're enrolled in, along with enrollment date and final grade.

Step 1: Create the Query

Create ➔ Query Design ➔ switch to SQL View ➔ insert the following SQL request:

```
SELECT  Student.StudentID, Student.FirstName, Student.LastName,
   Course.CourseName, Enrollment.Edate, Enrollment.FinalGrade
FROM Student, Enrollment, Course
WHERE Student.StudentID = Enrollment.StudentID
        AND Enrollment.CourseID = Course.CourseID ;
```

Save the query as **qry_StudentEnrollment**

We obtain the following resut: **TO MODIFY**

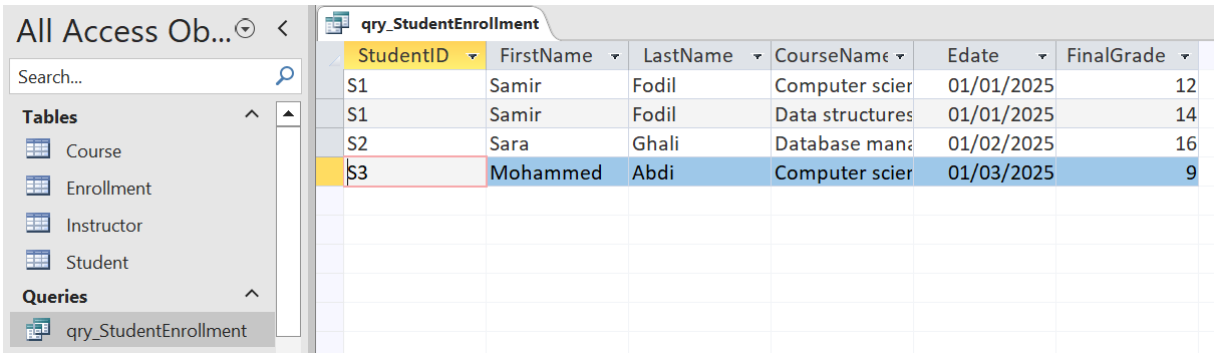


Figure 22 Result of the query

Step 2: Create the Report

1. Go to **Create** > **Report Wizard**.
2. Choose the query: qry_StudentEnrollment.
3. Select fields to include in the report:
    - StudentID
    - FirstName
    - LastName
    - CourseName
    - EnrollmentDate
    - FinalGrade
4. **Group by**: StudentID (optional but recommended to organize by student).
5. **Sort by**: CourseName or EnrollmentDate (optional).
6. Choose a layout: Tabular or Stepped works well.
7. Name the report: **rpt_StudentEnrollment**
8. You can switch to design view if you want to do some modiifcations or adjustments on the report.

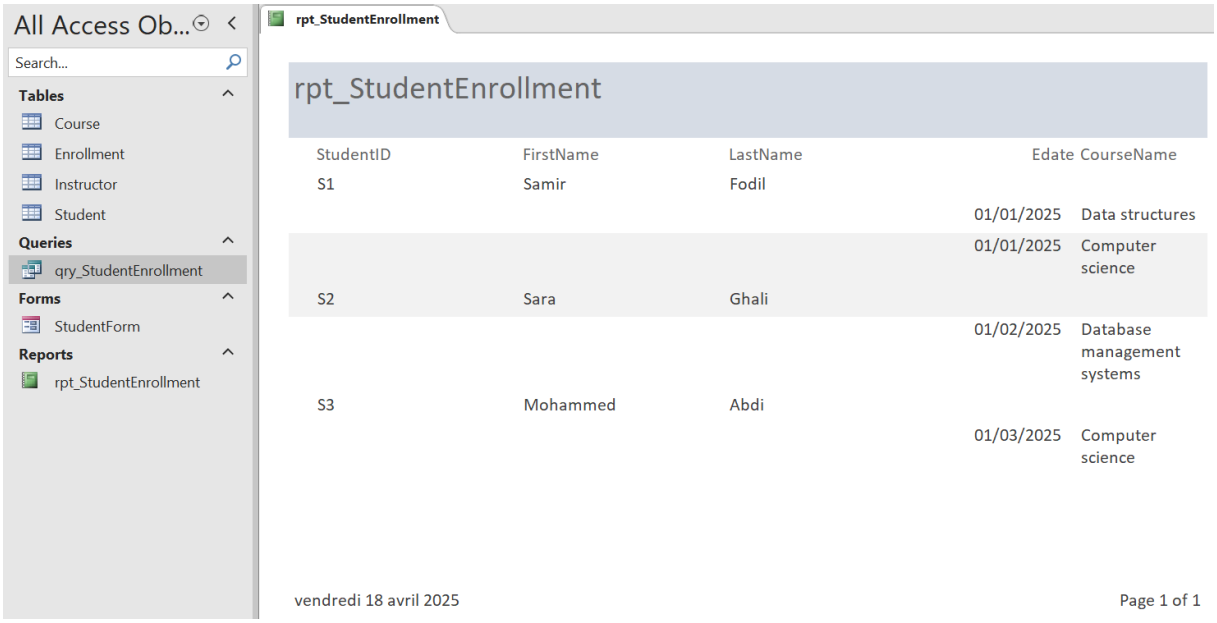


Figure 23 Creating the student enrollment report

5.8 USING MACROS

A macro is a tool that enables you to automate tasks and add functionality to your forms, reports, and controls. For example, if you add a command button to a form, you associate the button's **OnClick** event property to a macro that contains the commands that you want the button to perform each time that it is clicked.

It is helpful to think of Access macros as a simplified programming language in which you create code by building a list of actions to perform. When you build a macro, you select each action from a drop-down list and then fill in the required information for each action. Macros enable you to add functionality to forms, reports, and controls without writing code in a VBA module. Macros provide a subset of the commands that are available in VBA, and most people find it easier to build a macro than to write VBA code.

- ✓ Macros are sets of **actions** that automate tasks (e.g., opening forms, validating input, creating messages).
- ✓ Unlike VBA, macros are **easier to use** and **don't require programming skills**.

We can use macros to:

- Open a form automatically.
- Show a message when a button is clicked.

- Hide fields based on user input.

Creating Your First Macro:

Step1: On the **Create** tab, in the **Macros & Code** group, click **Macro**. This action will display the Macro Builder which is shown in the following illustration:

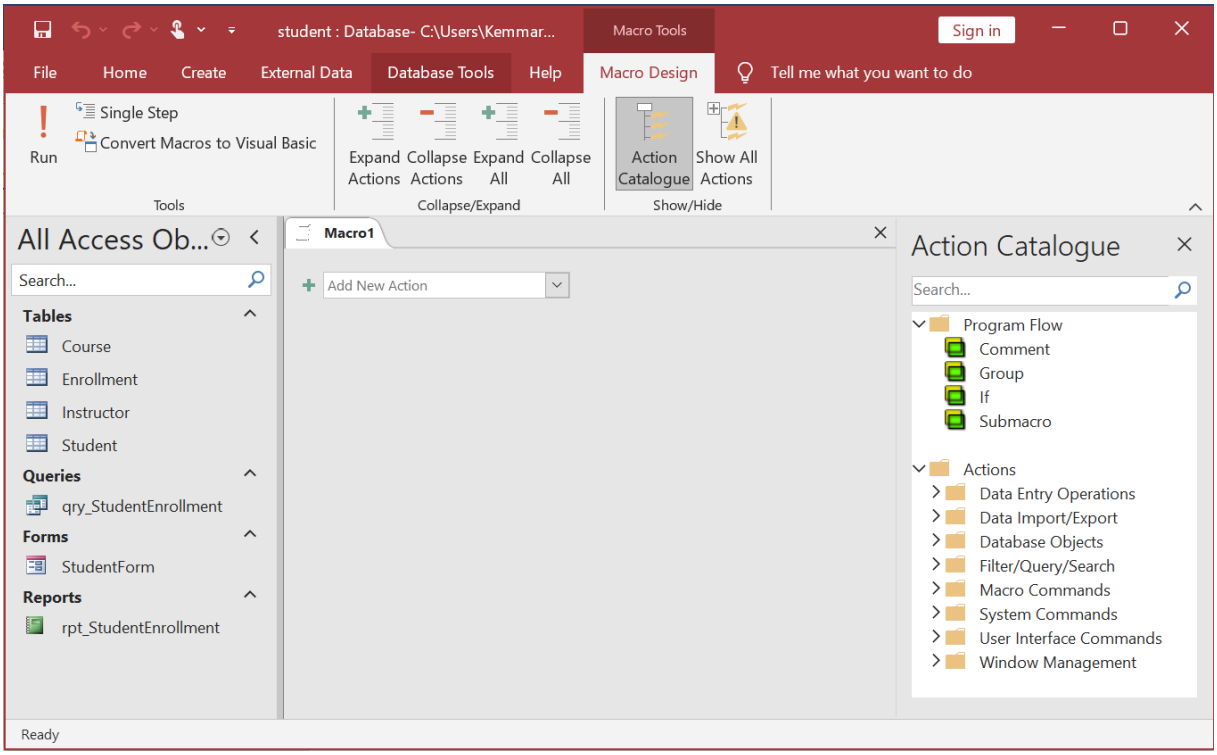


Figure 24 Creating a macro

Step 2: Add Actions to Your Macro

- In the Macro Builder, you'll see a drop-down labeled Add New Action.
- Click the drop-down arrow, and select an action from the list (for example: OpenForm, OpenReport, SetValue, etc.).
- After selecting an action, additional arguments (parameters) may appear below it.
    - For instance, if you chose OpenForm, you'll need to specify the Form Name, View, and optionally Filter Name or Where Condition.

Step 3: Configure the Action Arguments

- Fill in the necessary details for the selected action.
- For example, to open a form named CustomerForm, you would:
    - Action: OpenForm

- Form Name: CustomerForm
- View: Form
- Data Mode: Edit (or as needed)

Step 4: Add More Actions (Optional)

- If your macro needs to perform multiple steps, click Add New Action again to insert another action below the first.
- Repeat the selection and configuration for each action.

Step 5: Save the Macro

- Click the Save button (floppy disk icon) on the Quick Access Toolbar.
- Enter a name for your macro (e.g., OpenCustomerFormMacro).
- Click OK.

Step 6: Run the Macro

You can run the macro in several ways:

- From the Macro Builder, click Run (green play icon).
- Or, close the builder and run the macro from the Navigation Pane by double-clicking its name.
- Alternatively, assign the macro to a form button using a control's OnClick event in the Property Sheet.

Step 7 (Optional): Attach the Macro to a Form or Report

- Open the form/report in Design View.
- Select a button or event (e.g., On Load, On Click).
- In the Property Sheet, click the Event tab.
- Choose the macro you created from the drop-down list next to the event.

5.9 How to Secure Your Database in Access 2016

Method1: Set a Password and Encrypt the Database

- Open your database in **Exclusive Mode**:

- File → Open → Browse → Select your database → Click the dropdown arrow next to **Open** → Choose **Open Exclusive**.

- Then go to:
  - **File → Info → Encrypt with Password**.
  - Enter a strong password and confirm.

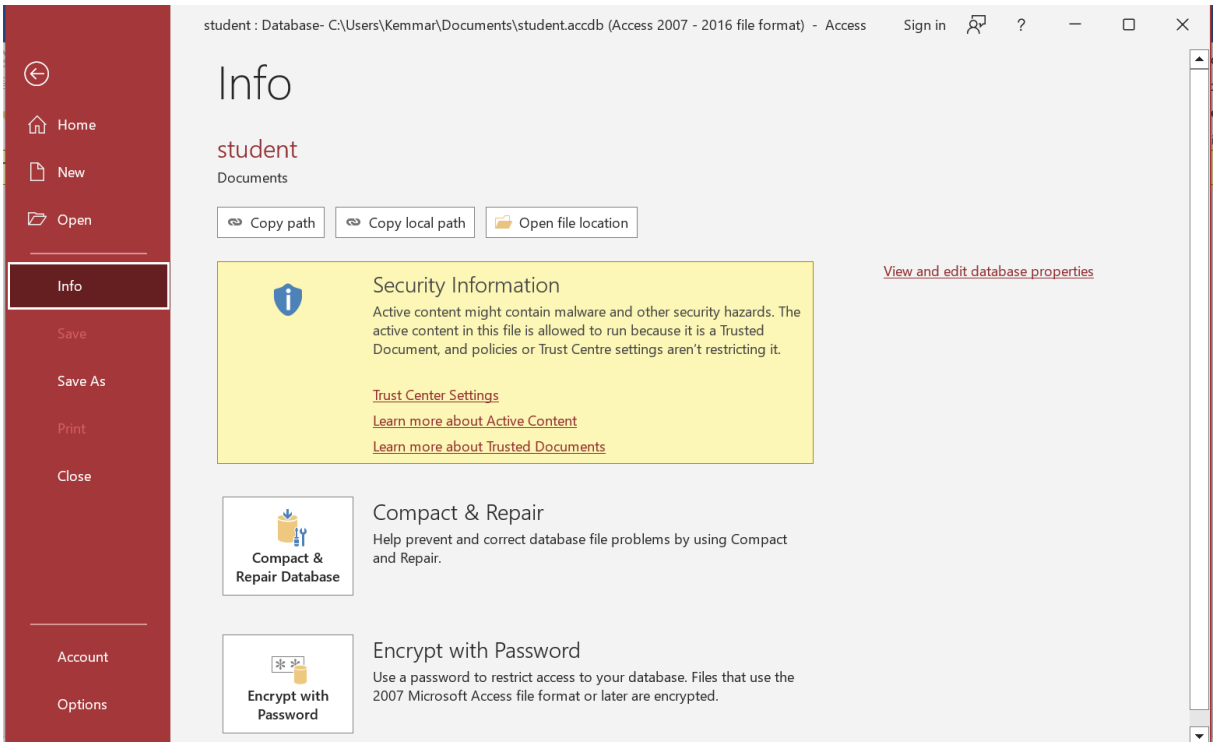


Figure 25 How to secure your database

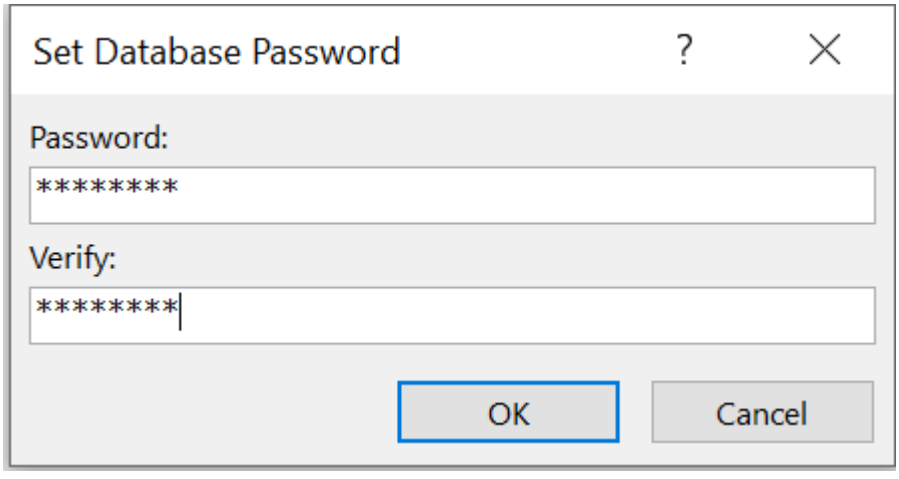


Figure 26 Choosing a password

- ✓ This encrypts the database file and prompts for the password every time it's opened.

Method 2: Split the Database (Frontend/Backend Architecture)

- Use the **Database Splitter** tool:
  - **Database Tools → Access Database** (Split Database).
- This separates:
  - **Backend (student_be.accdb)** – contains only tables.
  - **Frontend (student_fe.accdb)** – contains forms, queries, reports, macros, and links to the backend.
- ✓ Store the backend in a **secure network folder** with limited user access.

Method 3: Convert the Frontend to an ACCDE File

- Create a compiled version of your frontend:
  - **File → Save As → Make ACCDE**.
- This:
  - Locks the VBA code.
  - Prevents users from editing forms, reports, or modules.
- ✓ Helps protect your logic and user interface from tampering.

Method 4: Use File and Folder Permissions (Windows Security)

- Restrict access to folders where the database is stored:
  - Right-click the folder → **Properties** → **Security** tab.
  - Set read/write/modify permissions only for authorized users.
- ✓ Prevents unauthorized users from opening or modifying the database file.

Method 5: Hide Navigation Pane and Disable Special Keys

- Hide the Navigation Pane:
    - File → Options → **Current Database** → Uncheck **Display Navigation Pane**.
- Disable the Shift key bypass:
    - Use VBA code to prevent users from bypassing startup options by holding Shift.

5.10 CONCLUSION

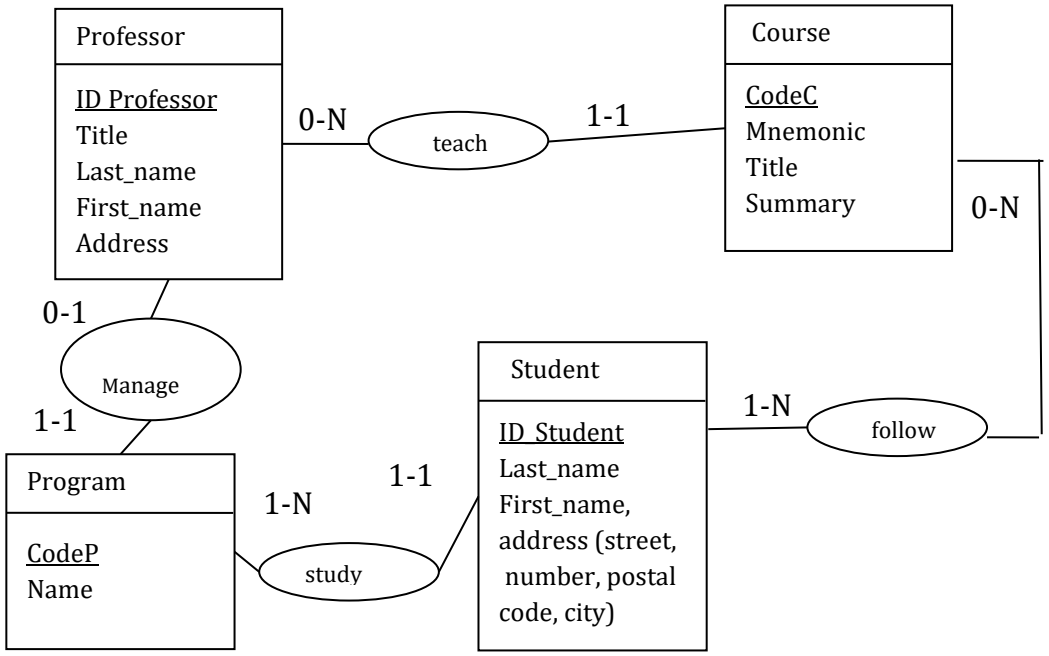
Microsoft Access is a versatile and powerful database management system that caters to users who need to store, manage, and analyze data in an easy-to-use environment. Whether you are building a database for a small business, academic purposes, or a personal project, Access provides a comprehensive set of tools to create and maintain effective databases.

CHAPTER 6

EXERCISE SOLUTIONS

6 EXERCISE SOLUTIONS

6.1 EXERCISE 2.6.1:



6.2 EXERCISE 2.6.7

1 Entities:

1. **Book**
    - Attributes:
        - Book_ID (Primary Key)
        - Title
        - ISBN
        - Publication_Year
2. **Author**
    - Attributes:

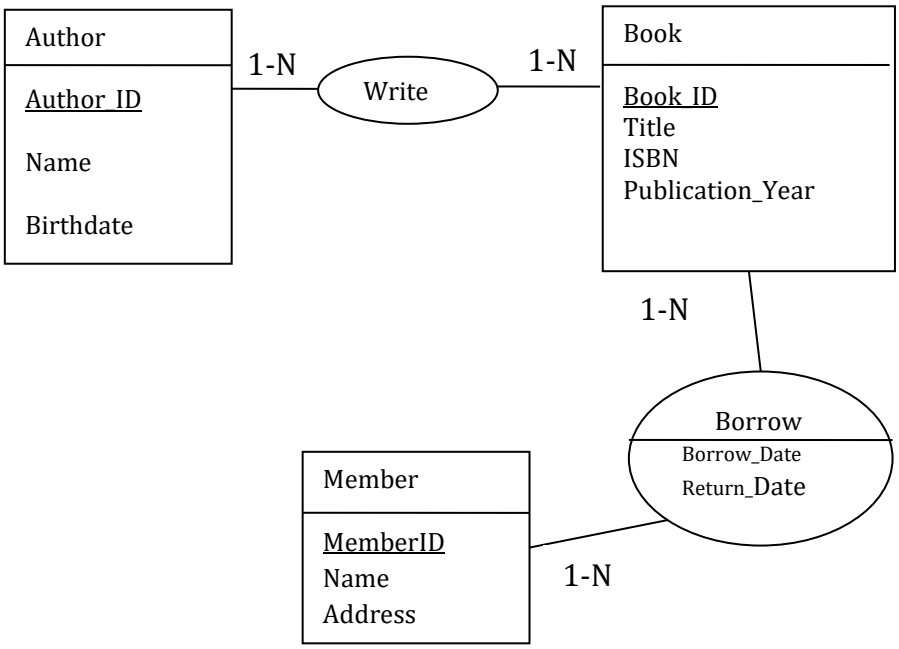
        - Author_ID (Primary Key)
        - Name
        - Birthdate

3. **Member**
    - Attributes:
        - Member_ID (Primary Key)
        - Name
        - Address

2 Relationships and Cardinality:

- **Author writes Book**
    - Cardinality: **Many-to-Many**
    - An **author** can write **many books**, and a **book** can have **multiple authors**.
    - This requires an intermediary relationship table: **Author_Book**.
    - **Attributes of Author_Book**:
        - Author_ID (Foreign Key referencing Author)
        - Book_ID (Foreign Key referencing Book)
- **Member borrows Book**
    - Cardinality: **Many-to-Many**
    - A **member** can borrow **many books**, and a **book** can be borrowed by **multiple members** (but one copy of each book at a time).
    - This is represented by the **Borrow** entity (which also contains the **Borrow_Date** and **Return_Date**).

3 ER Diagram:



4 Explanation of Diagram:

- **Author** and **Book** are related through the **Writes** relationship (many-to-many, so we use an intermediary **Author_Book** table).
- **Member** and **Book** are related through the **Borrows** relationship (many-to-many, stored in the **Borrow** table with additional attributes).

6.3 EXERCISE 2.6.3:

1. Identify the Entities and Their Attributes

Entities:

1. **Child**
    - **Attributes**: Child_ID (Primary Key), Name, Date_of_Birth, Guardian_ID (Foreign Key to Guardian), Health_Info
    - **Description**: Represents a child in the nursery. The **Guardian_ID** is a foreign key linking to the **Guardian** entity.
2. **Caregiver**
    - **Attributes**: Caregiver_ID (Primary Key), Name, Age, Qualification, Shift_Schedule
    - **Description**: Represents the caregiver in the nursery responsible for taking care of children

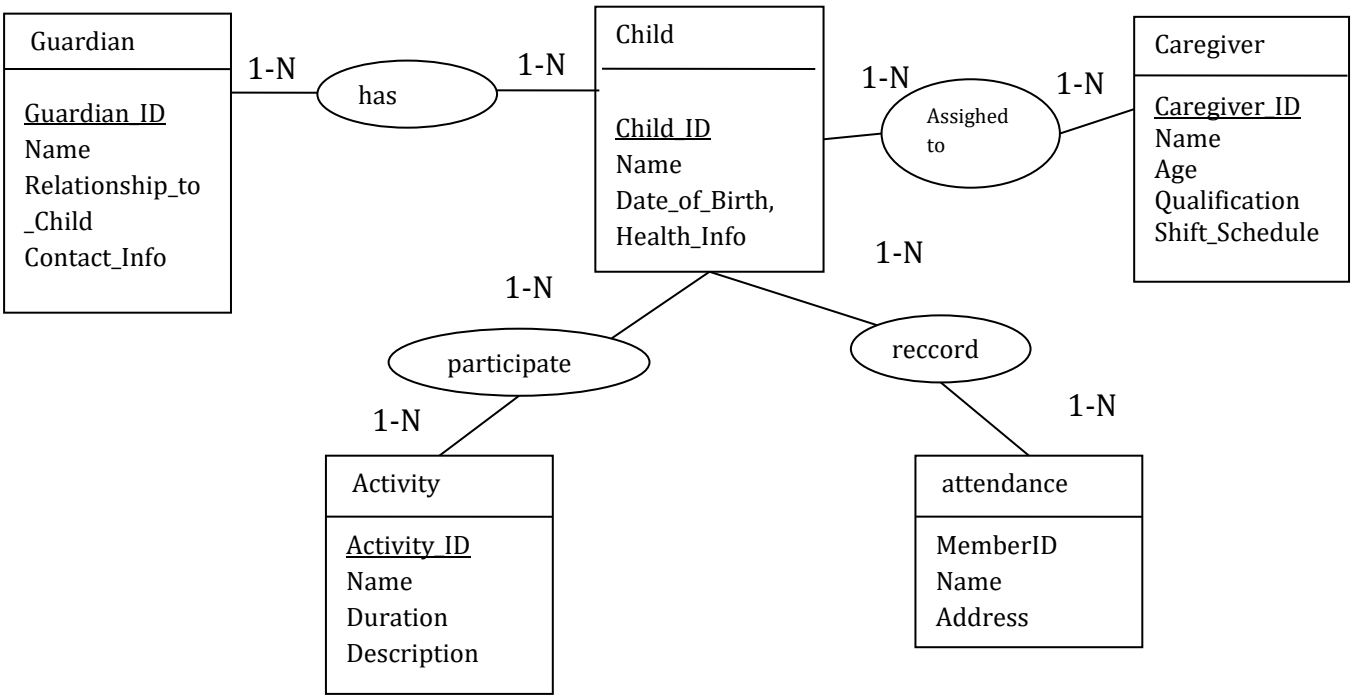
3. **Activity**
    - **Attributes**: Activity_ID (Primary Key), Name, Duration, Description
    - **Description**: Represents an activity organized in the nursery (e.g., playtime, learning sessions, meals).
4. **Attendance**
    - **Attributes**: Attendance_ID (Primary Key), Date, Status (Present/Absent), Child_ID (Foreign Key to Child)
    - **Description**: Represents the attendance record for each child on each day they attend the nursery.
5. **Guardian**
    - **Attributes**: Guardian_ID (Primary Key), Name, Relationship_to_Child, Contact_Info
    - **Description**: Represents the guardian of the child (parent or legal guardian).

2. Identify the Relationships

1. **Participates in** (Between **Child** and **Activity**):
    - **Cardinality**: Many-to-Many (A child can participate in multiple activities, and each activity can have multiple children).
    - **Solution**: Create a junction table called **Child_Activity** to store the participation.
2. **Assigned to** (Between **Caregiver** and **Child**):
    - **Cardinality**: One-to-Many (A caregiver can be assigned to multiple children, but each child is assigned to one caregiver).
3. **Records Attendance** (Between **Child** and **Attendance**):
    - **Cardinality**: One-to-Many (Each child has multiple attendance records, but each record belongs to one child).
4. **Has Guardian** (Between **Child** and **Guardian**):

- **Cardinality**: Many-to-One (A child can have one or more guardians, and each guardian can have one or more children).

3. ER Diagram



6.4 EXERCISE 3.6.1

A) The relational Model corresponding to Exercise 2.6.1:

Professor(ID Professor, Title, Last_name, First_name Address)

Student(ID_Student, Last_name, First_name, address, CodeP)

Program(CodeP, Name, ID_Professor)

Course(CodeC, Mnemonic, Title, Summary, ID_Professor)

Follow(ID_Student, CodeC)

B) The relational Model corresponding to Exercise 2.6.2:

Book(Book_ID, Title, ISBN, Publication_Year)

Author(Author_ID, Name, Birthdate)

Member(Member_ID, Name, Address)

C) **The relational Model corresponding to Exercise 2.6.3:**

Child(Child_ID, Name, Date_of_Birth, Health_Info, Guardian_ID)

Caregiver (Caregiver_ID, Name, Age, Qualification, Shift_Schedule)

Activity (Activity_ID, Name, Duration, Description)

Attendance (Attendance_ID, Date, Status (Present/Absent), Child_ID)

Guardian (Guardian_ID, Name, Relationship_to_Child, Contact_Info)

6.5 EXERCISE 3.6.2

Question 1:

(a) No, because the maximum cardinality of the "is" association for the *Vehicle* entity is 1. This means that a vehicle can have at most **one** model.

(b) No, because the minimum cardinality of the "owns" association for the *Owner* entity is 0.

(c) Yes, because the maximum cardinality of the "uses" association for the *Vehicle* entity is N.

(d) No, because each model uses **at least one** model (the minimum cardinality of the *used* association for the *Model* entity is 1).

Question 2:

The relational model is as follows:

- **Fuel**(*Designation*, Price, Pollution)
- **Model**(*Id*, CrashTest, Consumption)
- **Vehicle**(*RegistrationNumber*, Kilometers, ModelId, OwnerNumber)
- **Owner**(*Number*, LastName, FirstName, Age)
- **Uses**(*Designation*, Id)

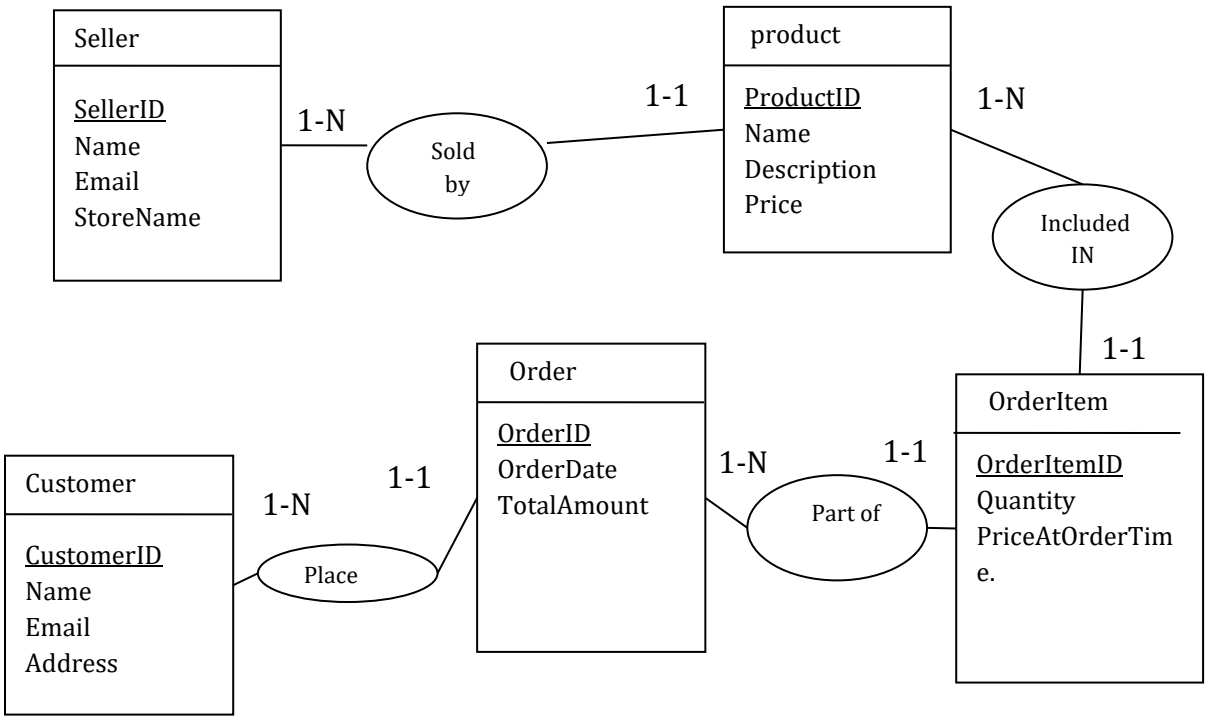
6.6 EXERCISE 3.6.3

Part1: Identified Entities and Relationships

Entities:

- **Product**
- **Seller**
- **Customer**
- **Order**

ER Diagram:



Part 2: Relational Schema

Using standard notation with **underlined primary keys** and **foreign key indications**:

Seller(SellerID, Name, Email, StoreName)

Product(ProductID, Name, Description, Price, SellerID)

Customer(CustomerID, Name, Email, Address)

Order(OrderID, CustomerID, OrderDate, TotalAmount)

OrderItem(OrderItemID,OrderID, ProductID, Quantity, PriceAtOrderTime)

N.B:

- ✓ OrderItem is a **junction table** for the many-to-many between Order and Product.
- ✓ Composite primary key: (OrderID, ProductID).

6.7 EXERCISE 4.4.1

1) CREATE TABLE Student (
   id VARCHAR(20) PRIMARY KEY,
   LastName VARCHAR(50) NOT NULL,
   FirstName VARCHAR(50) NOT NULL,
   age INT NOT NULL
   );
   CREATE TABLE Session(
   courseCode VARCHAR(20) NOT NULL,
   sessionNumber VARCHAR(50) NOT NULL,
   date DATE NOT NULL,
   room VARCHAR(10) NOT NULL,
   StartTime TIME NOT NULL,
   endTime TIME NOT NULL CHECK(endTimer > startTime),
   teacherID VARCHAR(20) NOT NULL,
   FOREIGN KEY (courseCode) REFERENCES Cours(courseCode),
   FOREIGN KEY (teacherID) REFERENCES Enseignant(teacherID),
   PRIMARY KEY (courseCode,studentID));

2) INSERT INTO teacher(teacherID,lastName,firstName) VALUES ('INFO2',
   'Houcine' , 'Mohammed');

3) INSERT INTO enrollment VALUES ("l0372","ALG2");

4) SELECT * FROM Student;

5) SELECT FirstName, LastName FROM Student WHERE age < 20;

6) SELECT lastName, firstName
FROM teacher, Course
WHERE teacher.teacherID = Cours.teacherID AND title="Statistics";

7) SELECT e.LastName, e.FirstName

FROM student e, enrollment i, course c

WHERE e.id= i.id AND i.code = c.code

AND (c.title="Probability" OR c.title=" Statistics ");

8) SELECT *

FROM student

WHERE age=(select min(age) from student);

9) SELECT count(DISTINCT teacher.teacherID)

FROM Session, Course

WHERE Session.courseCode = Course.courseCode AND title= " Probability";

10) SELECT date, room, startTime, endTime

FROM Session, Course

WHERE Session.courseCode = Cours. courseCode AND number = 1 AND title=" accounting ";

11) SELECT sessionNumber, date, room, startTime, endTime, e.nom, e.firstName

FROM Session, Course, teacher e

WHERE session.courseCode= Course.courseCode AND teacher.teacherID= session. teacherID

AND title ="Statistics" ORDER BY date, startTime;

12) UPDATE teacher SET rank="Maître-de-conférence B"

WHERE lastName="Senouci" AND firstName="Zaki";

13)INSERT INTO ‘session‘ VALUES ((SELECT code FROM Course WHERE title=’accounting’), (SELECT numberOfSession+1 FROM Course WHERE title="Logic"), ’2008-12-14’, ’S250’, 14:00’, ’18:00’, (SELECT teacherID FROM ‘teacher‘ WHERE lastName="Senouci" AND firstName= "Zaki"));

UPDATE course SET numberOfSession = numberOfSession +1 WHERE title= ’accounting’;

14) DELETE FROM enrollment WHRE studentID=’l0372’ AND code=’ALG2’ ;

6.8 EXERCISE 4.4.2

1. **Customers who placed at least one order**

```
SELECT DISTINCT c.Name, c.Email
FROM Customer c
JOIN Order o ON c.CustomerID = o.CustomerID;
```

2. **Product names and quantities in Order 102**

```
SELECT p.Name, oi.Quantity
FROM OrderItem oi, Product p
```

```
WHERE oi.ProductID = p.ProductID
 AND oi.OrderID = 102;
```

3. **Products never ordered**

```
SELECT Name
FROM Product
WHERE ProductID NOT IN (
   SELECT ProductID
   FROM OrderItem
);
```

4. **Total sales for each product**

```
SELECT p.ProductID, p.Name, SUM(oi.Quantity * oi.PriceAtOrderTime) AS TotalSales
FROM Product p, OrderItem oi WHERE
p.ProductID = oi.ProductID
GROUP BY p.ProductID, p.Name;
```

5. **Total revenue for each seller**

```
SELECT s.SellerID, s.Name, SUM(oi.Quantity * oi.PriceAtOrderTime) AS Revenue
FROM Seller s, Product p, OrderItem oi
WHERE (s.SellerID = p.SellerID
AND p.ProductID = oi.ProductID)
GROUP BY s.SellerID, s.Name;
```

6.9 EXERCISE 4.4.3

Let us take the following relational model:

Fuel(Designation, Price, Pollution)

Model(Id, CrashTest, Consumption)

Vehicle(RegistrationNumber, Kilometers, ModelId, OwnerNumber)

Owner(Number, LastName, FirstName, Age)

Uses(Designation, ModelId)

A) Basic Queries

1. **List all fuel types along with their price and pollution level.**

   SELECT Designation, Price, Pollution

   FROM Fuel;

2. **Show all vehicle registration numbers and the number of kilometers they've traveled.**

   SELECT RegistrationNumber, Kilometers

   FROM Vehicle;

B) Join Queries

3. **List the models along with the fuel they use.**

   SELECT m.Id, f.Designation, f.Price, f.Pollution

   FROM Model m

   JOIN Uses u ON m.Id = u.ModelId

   JOIN Fuel f ON u.Designation = f.Designation;

4. **List all vehicles with their owner's name and the fuel type they use.**

SELECT v.RegistrationNumber, o.FirstName, o.LastName, f.Designation AS Fuel

FROM Vehicle v

```
JOIN Owner o ON v.OwnerNumber = o.Number

JOIN Model m ON v.ModelId = m.Id

JOIN Uses u ON m.Id = u.ModelId

JOIN Fuel f ON u.Designation = f.Designation;
```

C) Conditions and Filtering

5. **Find all models that use fuel types with pollution greater than 120.**

```
SELECT DISTINCT m.Id

FROM Model m

JOIN Uses u ON m.Id = u.ModelId

JOIN Fuel f ON u.Designation = f.Designation

WHERE f.Pollution > 120;
```

6. **Find all owners under 30 years old who own at least one vehicle.**

```
SELECT DISTINCT o.FirstName, o.LastName

FROM Owner o

JOIN Vehicle v ON o.Number = v.OwnerNumber

WHERE o.Age < 30;
```

D) Aggregation

7. **Count how many vehicles use each fuel type.**

```
SELECT f.Designation, COUNT(*) AS VehicleCount

FROM Fuel f

JOIN Uses u ON f.Designation = u.Designation

JOIN Model m ON u.ModelId = m.Id

JOIN Vehicle v ON m.Id = v.ModelId
```

```
GROUP BY f.Designation;
```

8. **Find the average consumption of models using 'Diesel'.**

```
SELECT AVG(m.Consumption) AS AvgConsumption

FROM Model m

JOIN Uses u ON m.Id = u.ModelId

WHERE u.Designation = 'Diesel';
```

E) Subqueries

9. **List owners who own a vehicle that uses the least polluting fuel.**

```
SELECT DISTINCT o.FirstName, o.LastName

FROM Owner o

JOIN Vehicle v ON o.Number = v.OwnerNumber

JOIN Model m ON v.ModelId = m.Id

JOIN Uses u ON m.Id = u.ModelId

JOIN Fuel f ON u.Designation = f.Designation

WHERE f.Pollution = (

  SELECT MIN(Pollution) FROM Fuel

);
```

10. **Find vehicles whose models failed the crash test and use fuel costing more than 2.00.**

```
SELECT v.RegistrationNumber

FROM Vehicle v

JOIN Model m ON v.ModelId = m.Id
```

```
JOIN Uses u ON m.Id = u.ModelId

JOIN Fuel f ON u.Designation = f.Designation

WHERE m.CrashTest = 'Failed' AND f.Price > 46 ;
```

BIBLIOGRAPHY

[1]

Audibert, Laurent. *Cours-BD*. n.d. <https://laurent-audibert.developpez.com/Cours-BD/?page=bases-de-donnees-relationnelles>.

[2] Beaulieu, Alan. *Learning SQL Generate, Manipulate, and Retrieve Data*. 2020.

[3] Claude Chrisment, Karen Pinel Sauvagnat, Olivier Teste, Michel Tuffery. *Bases de données relationnelles*. EYROLLES, 2008.

[4] *cours-gratuits*. n.d. <https://www.cours-gratuit.com/cours-informatique/telecharger-cours-systeme-d-information-pdf>.

[5] emmanuel.coquery. n.d. <https://perso.liris.cnrs.fr/emmanuel.coquery/dokuwiki/lib/exe/fetch.php?media=enseignement:bd:lif4-td5-correction.pdf>.

[6] Gardarin, Georges. *Bases de donnees*. Eyrolles, 2003.

[7] Inisan, Hervé. *Access 2007 Référence*. Eyrolles , 2007.

[8] Jollois, François-Xavier. *https://fxjollois.github.io/cours-sql/*. n.d.

[9] Jouve, Mokrane Bouzeghoub and Mireille. *Le modèle relationnel*. Hermes, 1998.

[10]Sparfel, Céline. *Access 2010*. Micro application, 2010.

[11]TailoredRead. *Relational Databases - Your Custom-Tailored Book*. 2025.

[12]TAYLOR, Allen G. *SQL pour les Nuls*. 2014.