Database

2nd Preparatory classes

Prepared by: Dr. Mahmoud Chaira

Academic year: 2024-2025

Contents

Course objectives

At the end of this course, the student should be able to:

- Familiarize students with the concepts of databases, the evolution of Database Management Systems (SGBDs), and their fundamental goals.
- Understand how to design a database (Entity-Relaioship Model).
- Translate the E/R to the Relation Model (RM).
- Acquire knowledge and skills in using the ACCESS data manager.
- Automate certain data processing tasks with ACCESS tools.
- Acquire knowledge of the SQL language.

Prerequisites

No specific knowledge is required.

Chapitre 1: Introduction to information system

1.1 Concept of a System

A system is a set of interconnected and coordinated elements that interact to fulfill a specific mission. These elements may include human resources (staff), material resources (computers, machines), and procedural resources (rules and work methods).

Example: An inventory management system in a company. This system collects information about the number of products in stock, processes this data to determine critical levels, and disseminates alerts to managers when restocking is required. Inputs include customer orders, and outputs are actions like ordering new products or updating inventories.

1.2 Concept of an Organization

An organization is a structured entity where individuals collaborate to achieve specific goals. It can be commercial, administrative, or associative. The organization's role is to organize human, material, and immaterial resources to accomplish its missions.

Example: A Logistics company such as DHL. This organization uses an information system to track parcels, optimize delivery routes, and ensure communication between its different departments (operations, transport, customer service).

1.3. Definition of information systems

An Information System (IS) is a structured system designed to collect, store, manage, and disseminate information. It combines technology, people, and processes to support decision-making, coordination, control, analysis, and visualization within an organization. Information systems can vary significantly in complexity and scope, ranging from simple systems that manage a single department's data to large-scale enterprise systems that integrate information across multiple functions.

Example: Banking system allows banks to manage customer accounts, transactions, and loans. It processes millions of data points in real-time, enabling customers to check their balances online and managers to monitor financial flows.

1.4 Role of information systems in organizations

Information systems (IS) play a crucial role in organizations by supporting various functions and facilitating effective decision-making. Here are some key roles that information systems fulfill in organizations:

1. **Data Management**: Information systems help organizations collect, store, manage, and retrieve vast amounts of data efficiently. This data can include customer information, sales records, inventory levels, and financial data, allowing organizations to maintain organized records.
2. **Decision Support**: IS provides tools and analytical capabilities that assist management in making informed decisions. By processing and analyzing data, organizations can generate reports, dashboards, and visualizations that highlight trends, patterns, and key performance indicators (KPIs).
3. **Enhanced Communication and Collaboration**: Information systems facilitate communication and collaboration among employees, departments, and stakeholders. Tools such as email, instant messaging, and collaborative platforms (like Microsoft Teams or Slack) enable effective information sharing and coordination.
4. **Process Automation**: IS can automate routine tasks and business processes, improving efficiency and reducing manual effort. Automation of processes such as payroll, order processing, and inventory management allows organizations to save time and minimize errors.
5. **Customer Relationship Management (CRM)**: Information systems play a vital role in managing customer relationships by tracking interactions, sales history, and preferences. CRM systems help organizations enhance customer service, personalize marketing efforts, and build long-term relationships with clients.
6. **Operational Efficiency**: by integrating various functions such as finance, human resources, sales, and supply chain management, information systems streamline operations and eliminate silos. This leads to improved coordination, reduced redundancies, and enhanced overall efficiency.

1.5. Functions of the IS

The main functions of the IS are collecting, storing, processing, and disseminating information.

These steps are essential for efficiently managing information within the organization.

- **Collection**: The IS gathers data from various sources.
  - **Example**: In a supermarket chain, data from cash registers allows real-time tracking of sales for each product.
- **Storage**: Collected data is stored in databases.
  - **Example**: An insurance company stores claims in a database for future processing.
- **Processing**: The system analyzes the information to extract useful insights.
  - **Example 1**: Management wants to have a list of the best-selling products and the list of the least-sold products during a given period.

  - **Example 2**: In a factory, data on machine performance is analyzed to identify potential breakdowns.
- **Dissemination**: Processed information is sent to decision-makers or stakeholders.
  - **Example**: Performance reports generated by a management IS are sent to the executives to evaluate team productivity.

1.6 Aspects of the Information System (IS)

The Information System (I.S.) is composed of both static and dynamic aspects, which together ensure the smooth operation and adaptability of the system to meet an organization's evolving needs.

Static Aspect of the IS

The static aspect refers to the foundational, permanent components of the I.S., which typically include the hardware, software, databases, and infrastructure that support the system's operations. These components remain largely unchanged over time and provide the backbone on which the IS operates.

- **Databases**: Serve as repositories for storing large volumes of structured and unstructured data. They ensure data integrity, security, and easy retrieval.
- **Servers and Networks**: Facilitate the storage, sharing, and transmission of data across the organization.
- **Applications**: Enterprise-level software such as Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) tools, and human resources systems.

Example: In an ERP system, there are fixed modules designed to handle specific functions such as accounting, human resources, and inventory management. These modules are static because they are permanently installed and integrated into the organization's I.S. to manage specific operational tasks.

- **Accounting Module**: Manages financial records, processes transactions, and generates reports.
- **Human Resources Module**: Tracks employee records, payroll, attendance, and benefits.
- **Inventory Management Module**: Monitors stock levels, manages reordering, and tracks shipments.

These modules provide the organization with stability and structured processes that are crucial for daily operations. The static aspect ensures reliability and continuity, as these systems do not require constant change and provide the structure needed for business functions.

Dynamic Aspect of the IS

The dynamic aspect of the IS refers to the flexible, adaptable processes and workflows that evolve in response to the changing needs of the organization. Unlike the static components, the dynamic aspect is more fluid and involves real-time information flows, decision-making, and task automation.

Dynamic systems are often involved in:

- **Real-time Data Processing**: Handling live information from various sources and ensuring that decisions can be made instantly based on this data.
- **Automated Workflows**: These are adaptable processes that can change based on new inputs or demands. For example, how the system responds to fluctuating customer queries or real-time updates in supply chain information.

Example: A call center I.S. is a prime example of the dynamic aspect. The system manages the flow of incoming calls and dynamically adjusts by redirecting customers to agents based on their availability, skill set, and workload. As customer needs change, the IS adapts in real-time to provide the best customer service experience. If a higher number of calls come in during peak hours, the system automatically adjusts, perhaps by prioritizing certain calls or rerouting them to specialized teams.

- **Call Routing**: The system identifies the best-suited agent for the customer based on the query type, the agent's skills, and current availability.
- **Real-Time Updates**: If there are unexpected events like system outages or agent unavailability, the IS dynamically reroutes or queues calls to ensure minimal disruption.

Dynamic systems are crucial for supporting an organization's agility and responsiveness to both internal and external changes. By adapting to new demands or fluctuations in the business environment, the dynamic aspect of an IS helps an organization stay competitive and responsive.

1.7. Components of an Information System

The components of an IS are varied and include information, human resources, material resources, and methods.

- **Information**: These are the data that the IS manipulates.
  - **Example**: Customer data on an e-commerce website (name, address, purchase history) is crucial information for personalizing offers and improving customer experience.
- **Human resources**: These are the people who use and manage the IS
  - **Example**: In a bank, financial analysts use an IS to monitor transactions and assess financial risks.
- **Material resources**: These include computers, servers, networks, and other equipment.
  - **Example**: A data center stores millions of customer records, ensuring the availability of information 24/7.
- **Methods**: These are the defined processes for managing information flows.
  - **Example**: A hospital uses strict protocols for accessing electronic medical records, ensuring both data integrity and confidentiality.

Information systems play a vital role in modern organizations, providing the necessary tools to streamline operations, enhance communication, support strategic planning, and improve

customer service. By effectively managing information, organizations can gain a competitive advantage, make informed decisions, and respond swiftly to changing market conditions.

Chapter 2: Databases & DBMS

2.1 Database definition

A database is a collection of data that is organized in a manner that allows efficient retrieval, management, and modification of the data. The database is structured in a way that makes it easy to access and manipulate large amounts of information, typically through the use of a Database Management System (DBMS). The DBMS serves as an intermediary between the users and the database, ensuring data consistency, integrity, security, and performance while providing ways to query, update, and manage the stored data.

2.2 History of Database Management Systems (DBMS)

The history of Database Management Systems (DBMS) traces back to the early efforts of managing data efficiently in large-scale computing environments. The evolution of DBMS reflects advancements in computer hardware, software, and the increasing need for structured data storage and retrieval systems.

1. **1950s - Early Beginnings**: DBMS development can be traced back to the 1950s when magnetic tapes and punch cards were used for storing data. Data was processed in sequential batches, with no direct interaction between users and the data.
2. **1960s - File Systems & Hierarchical Databases**: In the 1960s, organizations began creating file-based systems. However, these systems lacked flexibility, as each application was tightly coupled with its own file structure. This led to the development of the first true DBMS models.
    - **Hierarchical Model**: IBM introduced the Information Management System (IMS) in 1966, which used a hierarchical structure to store data. Data was organized in a tree-like structure, with parent-child relationships. While efficient for certain applications, it lacked flexibility due to its rigid structure.
3. **1970s - The Rise of Relational Databases**: In 1970, Edgar F. Codd, a mathematician at IBM, proposed the **Relational Model**, which fundamentally changed how databases were designed. Unlike hierarchical databases, the relational model used tables (or relations) and allowed data to be accessed in a more flexible manner using Structured Query Language (SQL).
    - **SQL**: SQL, developed in the 1970s, became the standard language for managing and querying relational databases.

- **First Relational DBMS**: Oracle Corporation was among the pioneers to commercialize relational DBMS in the late 1970s.

4. **1980s - Commercialization of RDBMS**: The 1980s saw the wide adoption of Relational Database Management Systems (RDBMS). Several vendors, such as Oracle, IBM (with DB2), and Microsoft (with SQL Server), emerged as key players.
    - **Standardization**: ANSI and ISO standardized SQL in the mid-1980s, promoting its use in relational databases.
5. **1990s - Object-Oriented and Distributed Databases**: The 1990s introduced the concept of Object-Oriented Database Management Systems (OODBMS), which allowed databases to store complex data types and support object-oriented programming. However, they never gained the same popularity as relational databases.
    - **Distributed Databases**: As networks grew in complexity, distributed databases became more important, allowing data to be stored across multiple locations but managed as a single database.
6. **2000s - NoSQL and Big Data**: With the rise of the internet, social media, and big data, traditional RDBMS systems faced challenges in handling large volumes of unstructured data.
    - **NoSQL Databases**: Non-relational (NoSQL) databases, such as MongoDB, Cassandra, and CouchDB, emerged as alternatives to RDBMS. These databases were designed to handle high scalability, flexibility, and distributed environments.
    - **Big Data Frameworks**: The development of big data frameworks like Hadoop and Spark revolutionized how massive datasets were processed and managed.
7. **2010s - Cloud Databases & New Trends**: The 2010s witnessed the shift towards cloud-based databases and Database-as-a-Service (DBaaS). Major providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud introduced scalable, managed database services that allowed organizations to offload the complexity of database management.
    - **NewSQL**: NewSQL databases emerged as a hybrid solution, combining the scalability of NoSQL with the consistency of relational databases.
8. **Current Trends**: Today, DBMSs continue to evolve, with advances in AI integration, blockchain-based databases, and multi-model databases. These systems provide flexibility in handling both structured and unstructured data, enabling new applications such as real-time analytics and IoT data management.

The history of DBMS demonstrates the transition from simple file-based systems to sophisticated, highly scalable solutions, reflecting the growing complexity and scale of modern data-driven applications.

2.3 Advantages of DBMS

- **Data Integrity and Accuracy**: DBMS enforces data integrity constraints, ensuring that the data remains accurate and consistent over its lifecycle. It helps to prevent data anomalies through normalization and other integrity checks.

- **Data Security**: A DBMS provides various security measures to protect data from unauthorized access. This includes user authentication, access control, and encryption, ensuring that sensitive information is secure.

- **Data Redundancy Reduction**: By centralizing data storage, a DBMS minimizes data redundancy, reducing the likelihood of data duplication. This leads to more efficient use of storage space and ensures that data is updated in a single location.

- **Improved Data Access**: DBMSs facilitate efficient querying and retrieval of data using structured query languages (like SQL). Users can easily access the data they need without requiring extensive programming knowledge.

- **Backup and Recovery**: A DBMS provides robust backup and recovery solutions, enabling organizations to protect their data from loss due to hardware failures, system crashes, or other disasters. Regular backups ensure data can be restored to a previous state when needed.

- **Multi-User Support**: DBMSs allow multiple users to access and manipulate the database simultaneously while maintaining data consistency and integrity. This is crucial for collaborative environments where many users need to work with the same data.

Chapter 3: Entity-Relationship (ER) Model

3.1 Introduction to the ER Model

The Entity-Relationship (ER) Model is a foundational tool in database design that helps visualize and design the structure of a database in a way that maps out the relationships between the data elements. Proposed by Peter Chen in 1976, it is widely used in database design to define data elements (entities) and the relationships between them.

The main purpose of the ER model is to provide a high-level conceptual design of the database, which can later be translated into a relational schema.

3.2 Components of the ER Model

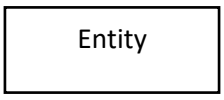
The ER model is based on three primary concepts: Entities, Attributes, and Relationships. These components help structure the data and how it interacts within the system.

3.2.1 Entities

An Entity, also known as entity type represents a real-world object or thing that exists on its own. Entities are things that we store data about. An entity could be a physical object, such as a student, vehicle, or employee. It could be something abstract, like a course, a job, or a bank account. In ER model, an entity is represented by a rectangle.

Entities can be classified into:

- **Strong Entities**: These entities have an independent existence in the database. They are represented by a rectangle in ER modeling as shown below.
    - Example: A *Student* entity can exist independently of other entities.

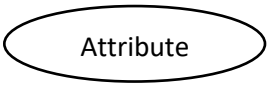


- **Weak Entities**: These entities depend on a strong entity for their existence. They cannot be uniquely identified by their attributes alone and rely on a relationship with a strong entity. They are depicted by a double rectangle.

- Example: A *Course Registration* entity depends on the *Student* and *Course* entities.

3.2.2 Attributes

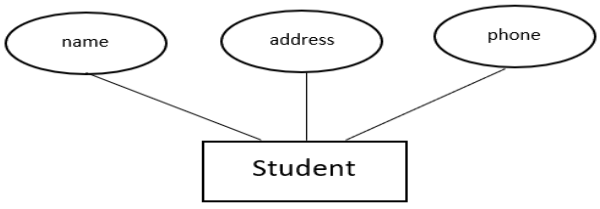
An Attribute represents the properties or characteristics of an entity. Each entity has multiple attributes that provide more information about it. In ER modeling, the notation used to represent a weak entity is shown below.



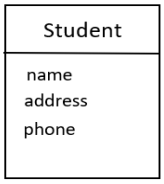
Attributes can be of different types:

- **Simple (Atomic) Attributes**: Attributes that cannot be divided into smaller parts.
  - **Example:** A *Student* entity may have a simple attribute called *StudentID*.
- **Composite Attributes**: Attributes that can be subdivided into smaller parts.
  - **Example:** A *Name* attribute can be subdivided into *FirstName* and *LastName*.
- **Derived Attributes**: Attributes that can be calculated or derived from other attributes.
  - **Example:** An attribute *Age* can be derived from *DateOfBirth*.
- **Multi-Valued Attributes**: Attributes that can hold multiple values. It is represented as
  - **Example:** A *PhoneNumber* attribute for a *Student* entity may store multiple phone numbers.

Attributes are depicted by ellipses connected to their respective entity in an ER diagram. The figure below shows an ER diagram featuring a single entity, "Student," along with three attributes: name**,** address**,** and phone**.**

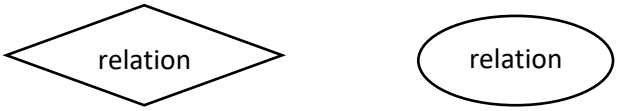


The concise form is easily created from the standard form, especially when space is an issue as shown in the figure below.



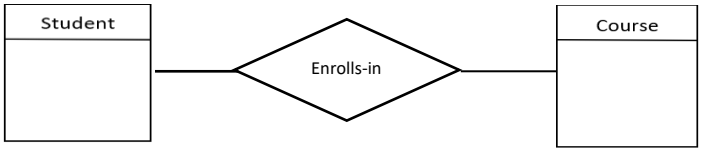
3.2.3 Relationships

A relationship describes how entities are related to one another. Relationships are used to connect two or more entities and specify how these entities interact. Relationships are usually denoted by verb phrases, and it is represented as a diamond as shown below or with an oval.



A relation can also be denoted with a circle.

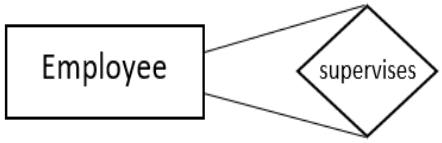
- **Example**: In a university database, a Student can enroll in a Course. Therefore, there is a relationship between the Student entity and the Course entity, called Enrollment.



Relationships have the following characteristics:

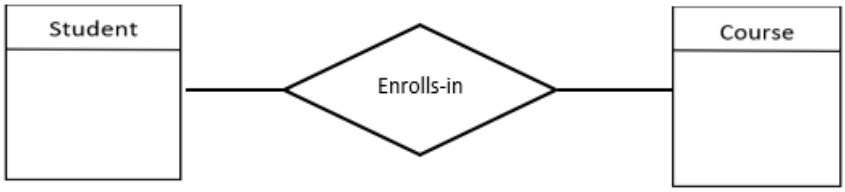
Degree of Relationships: This specifies the number of entities involved in a relationship. The n-ary relationship represents the general case for degree n, while specific instances include unary, binary, and ternary relationships, which have degrees of 1, 2, and 3, respectively.

- **Unary Relationship**: A relationship involving only one entity type, it is represented by a diamond, with both connections pointing to the same entity. (e.g., an employee supervises other employees).

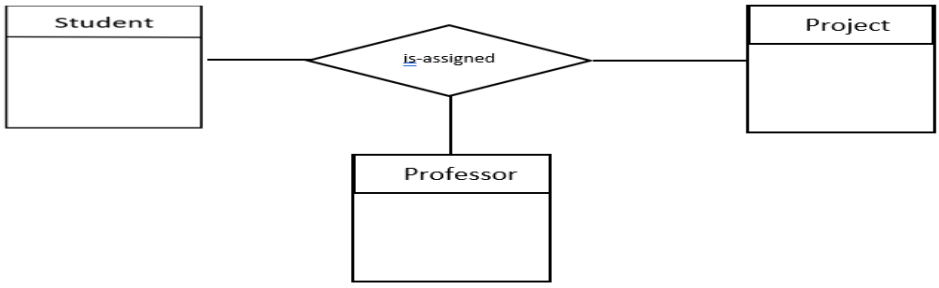


Note: In recursive relationships, it is important to specify the roles that each entity type plays (in our example, the roles would be supervisor and supervised).

- **Binary Relationship**: A relationship involving two entity types. It is the most common type found in the real world. In fact, numerous modeling systems employ only this type. (e.g., a Professor teaches a Course).



- **Ternary Relationship**: A relationship involving three entity types (e.g., a Student is assigned a Project by a Professor). A ternary relationship is necessary when binary relationships alone cannot effectively capture the meaning of the association. It is represented by a single diamond linked to three entities, as illustrated bellow.

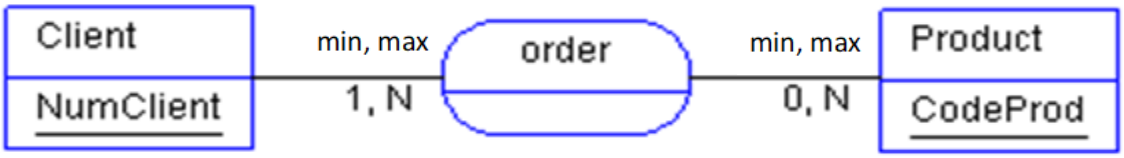


Cardinality of Relationships: A pair of cardinalities will be assigned to each entity belonging to an association. These cardinalities will specify how each entity participates in the association.

This pair of cardinalities consists of a minimum cardinality and a maximum cardinality. The value of these cardinalities will depend on the management rules of the users.

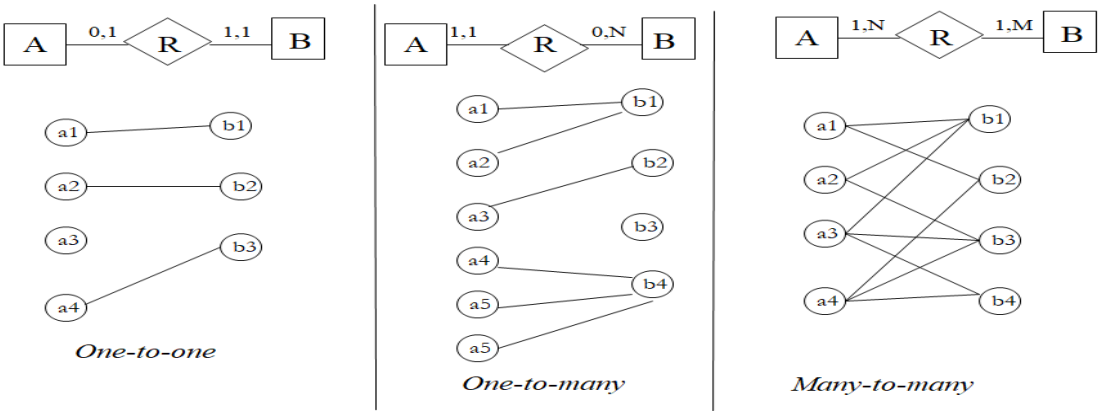
The minimum cardinality can be either 0 or 1, representing the minimum number of times an entity can use this association. The maximum cardinality can be either 1 or n, representing the maximum number of times an entity can use this association.

Example: a customer must order at least one product; a product can be ordered by zero or any number of customers.



Based on the maximum cardinalities, a binary association (degree = 2) can be of type 1-1, 1-N, or N-M as shown is the figure below:

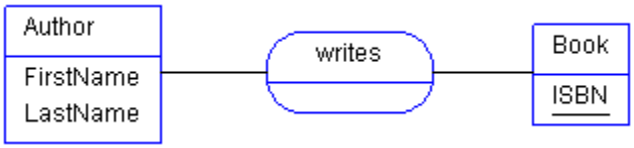
- **1-1**: Each instance in the first entity is associated with at most one instance in the second entity, and vice versa.
- **1-N**: Each instance in the first entity is associated with multiple instances in the second entity, but each instance in the second entity is associated with at most one instance in the first entity.
- **N-M**: instances in both entities can be associated with multiple instances in the other entity. This represents a many-to-many relationship.



Exercises

Exercise 1

Given the following incomplete Entity Relationship model (ER):



Complete the model with the following management rules:

- An author can write one or more books, and a book can be written by one or more authors.
- A book has a title and contains many pages.
- Each author has a nationality.
- The first name, last name, and nationality of the author are not enough to uniquely identify them.
- A book is published by one and only one publisher, and a publisher can publish one or more books.
- A publisher has a unique name, a phone number, and an address.

Exercise 2

A bank wishes to design a database to store information about its customers. It also wants to store the contact details of each customer (first name, last name, and address), the accounts they hold, and their balance (knowing that some accounts may have multiple owners). Additionally, transactions related to each account (withdrawals and deposits, along with their date and amount) will be stored.

1. Identify the entities and their attributes required to fulfill this requirement.
2. Specify the relationships between entities, including any properties (attributes) of these relationships.
3. Define the cardinalities for each association.

Exercise 3

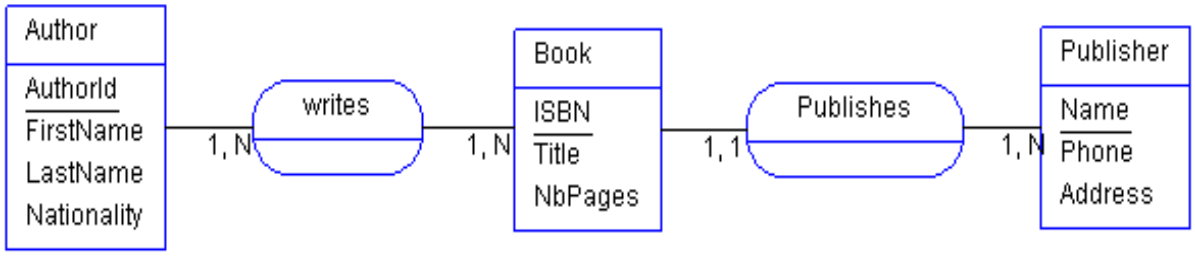
The goal is to model a database for a system that manages the sale of products to individuals.

The products in the store have a reference (a code), a label, and a unit price, while customers have an identity (name, surname, address). Customers place product orders.
The date of the order is recorded. For each order, the customer specifies a delivery address.
The order concerns a certain number of products, with a specified quantity for each product.

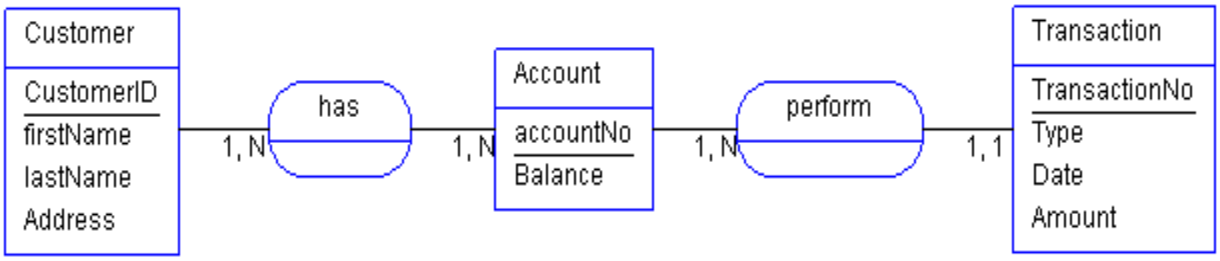
- Draw the Entity-relationship model.

Corrections

Exercise 1



Exercise 2



Chapter 4: Relational model

After completing the ER diagram, you can transform the conceptual schema into the relational model that your DBMS requires. In modern database systems, the majority of new installations are built on the relational data model, and databases that follow this model are referred to as relational databases.
A relational database is structured entirely as a collection of relations, not based on "relationships between files" as is sometimes mistakenly thought. This chapter will explain what a relation truly is and how it represents data relationships within the database.

1. Components of relational database

Attribute: An attribute is an identifier (a name) that describes information stored in a database. Examples of attributes include: a person's age, a person's name, and a social security number.

Domain: A domain represents the allowed set of values for a column, with each column drawing its values from one specific domain. This makes relations column-homogeneous. Additionally, every column must comply with a domain constraint. Depending on your DBMS, this constraint could be a simple data type like integers or dates, or it could allow you to define custom, detailed domains to apply to columns.

Example:

Social_security_numbers: The list of valid nine-digit numbers.

Names: The list of character strings that correspond to individuals' names.

Employee_ages: The valid age range for employees, which must be between 15 and 80 years.

Relation: A relation is a subset of the Cartesian product of n attribute domains ($n > 0$). A relation is represented in the form of 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 along with the list of attributes and their domains.

Example: Person(NSS: Integer, Last_Name: String, First_Name: String)

Primary key: A unique identifier consists of one or more columns that uniquely distinguish rows within a table. When the primary key includes multiple columns, it is also referred to as a "composite key."

Foreign Key: A foreign key is the primary key from another table that is referenced in the current table. It serves as the link to establish a relationship between the two tables and enables the database management system (DBMS) to enforce referential integrity.

2. Integrity constraints

Integrity constraints are rules applied to data within a database to ensure its accuracy, consistency, and validity. They restrict the types of data that can be stored, thereby maintaining the integrity of the database.

Types of Integrity Constraints

a. Primary Key Constraint: A primary key constraint uniquely identifies each record in a table, ensuring that no two rows have the same key value and no key value is null.

b. Domain Constraint: A domain constraint limits the acceptable values for an attribute, guaranteeing that only valid data types and specific value ranges can be entered. For example, in a database storing student information, a domain constraint on the "Age" attribute could specify that only integers between 18 and 65 are allowed. This ensures that the age entered is both a valid number and within the appropriate range.

c. Referential integrity: The relational data model enforces a rule known as referential integrity, which requires that every non-null foreign key value must correspond to an existing primary key value. This is likely the most crucial constraint in a relational database, as it guarantees the consistency of references between tables.

3. Mapping relationships to a relational database

In this section, we build upon the mapping rules introduced at the end of Chapter 4. In that chapter, we learned how to map entities, including those with composite and multivalued

attributes. Now, after discussing the structural constraints of relationships, we will explore how to map relationships.

To translate an Entity-Relationship model to the Relational Model, the following rules can be applied:

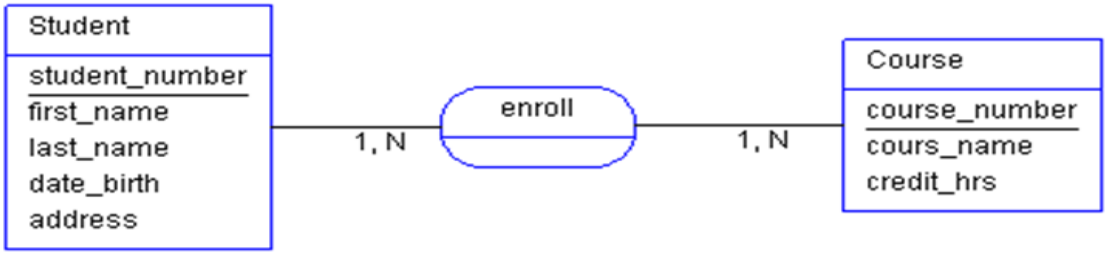
R1: Each Entity gives rise to a Relation (table). Each property of this Entity becomes an attribute of the Relation. The identifier is retained as the primary key of the relation.

Every entity is translated into a relation (table) where:

- The name of the table is the name of the entity,
- The primary key of the table is the identifier of the entity,
- The other attributes of the entity form the other columns of the table.

R2: Any binary (or n-ary) relationship, where the cardinalities of each participating entity are in the form (0,n) or (1,n), is translated into a new relation (table) where:

- The name of the table is the name of the association.
- The primary key of the table is formed by concatenating the identifiers of each of the entities participating in the relationship.
- The attributes of the association form the other columns of the table.

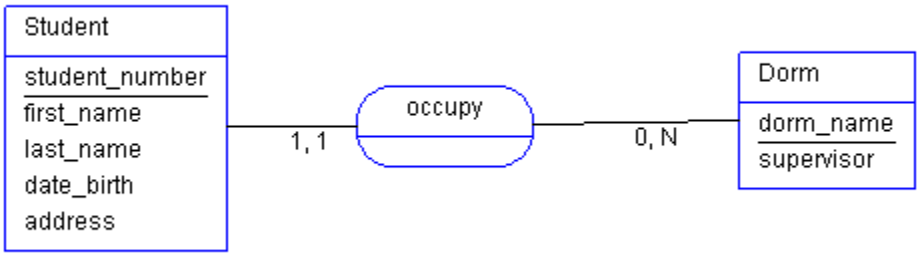


The corresponding relational model is as follows:

Student (<u>student_number</u>, first_name, last_name, date_birth, address)
Course (<u>course_number</u>, cours_name, credit_hrs)
enroll (#<u>student_number</u>, #<u>course_number</u>)

R3: Any binary relationship where the cardinalities of one entity are in the form (0,1) or (1,1) and the cardinalities of the other participating entity are in the form (0,n) or (1,n) is generally translated by a key reference:

- The identifier of the entity with cardinalities in the form (0,n) or (1,n) is added as an additional column to the table representing the other entity. This new column is called a foreign key denoted with #.



Students must live in only and only one dorm room, **and** Dorm rooms may have zero or more students.

The corresponding relational model is as follows:

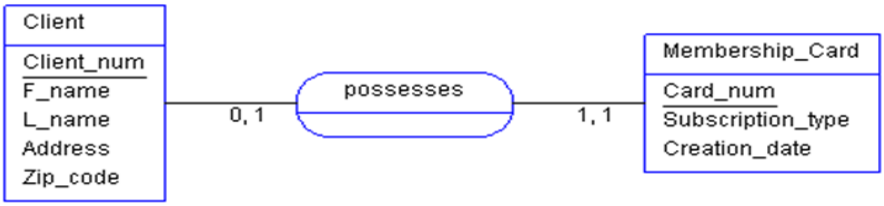
Student (student_number, first_name, last_name, date_birth, address, #dorm_name)
Dorm (dorm_name, supervisor)

R4: Given that a binary relation of the form (1,1)-(1,1) cannot exist, we are left with the following two scenarios:

1. Binary relation (0,1)-(1,1)
2. Binary relation (0,1)-(0,1)

Binary relation (0,1)-(1,1)

We duplicate the key of the table based on the entity with cardinality (0,1) in the table based on the entity with cardinality (1,1).



The Client_num, which is the primary key of the Client table, becomes a foreign key in the Membreship_Card table.

The corresponding relational model is as follows:

Client(Client_num, F_name, L_name, Address, Zip_code)
Membership_Cared(Card_num, Subscription_type, Creation_date, #Client_num)

Binary relation (0,1)-(0,1)

We duplicate the key from one of the tables into the other. If the relationship has its own properties, they also become attributes in the table where the foreign key is inserted.

The corresponding relational model is as follows:

Employee (Employee_num, F_name, L_name)
Company (Company_num, Headquarters_addr, #employee_num)

Or

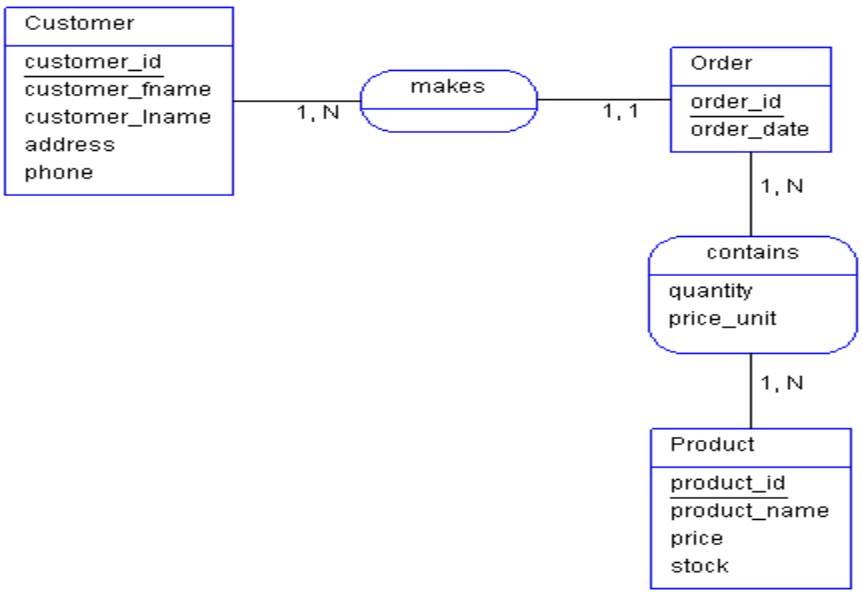
Employee (Employee_num, F_name, L_name, # Company_num)
Company (Company_num, Headquarters_addr_Company)

We can either transfer the primary key from the Company table to the Employee table, or we do the reverse.

Exercises

Exercise 1:

Convert the following Entity Relationship **(**ER) model into a Relational model**:**



Exercise 2:

Perform the reverse conversion (Relational model to ER model) of the following schema:

Department (Code, Name, Location, Address)
Employee (ID, LastName, FirstName, Position, Salary, #Department)
Project (Code, Name, Type, StartDate)
Participate (#Employee, #Project)

Exercise 3

Consider the following relational model for managing the annual grades of students:

STUDENT (StudentID, LastName, FirstName, Age)
MODULE (ModuleCode, ModuleName, Coefficient)

EVALUATE (StudentID, SubjectCode, Date, Grade)

Questions:

1. Identify the keys in the schema above.
2. Can a student have multiple grades in the same module?
3. Define integrity constraints for the EVALUATE relation.
4. Perform the reverse conversion back to the ER model.

Corrections

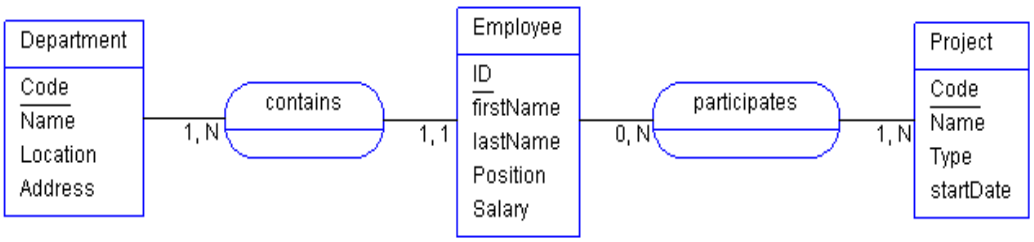
Exercise 1

CUSTOMER (custemer_id, customer_fn, customer_ln, address, phone)
ORDER (order_id, order_date)
PRODUCT (product_id, product_name, price, stock)
CONTAINS (#order_id, #product_id, quantity)

Primary keys are underlined.

Foreign keys are prefixed with #

Exercise 2



Exercise 3

1.

STUDENT (StudentID, FirstName, LastName, Age)
COURSE (CourseID, Name, Coefficient)
EVALUATE (#StudentID, #CourseID, Date, Grade)

Primary key: Underlined

Foreign key: Prefixed with #

Composite key: A primary key made up of multiple attributes (e.g., StudentID + CourseID + Date).

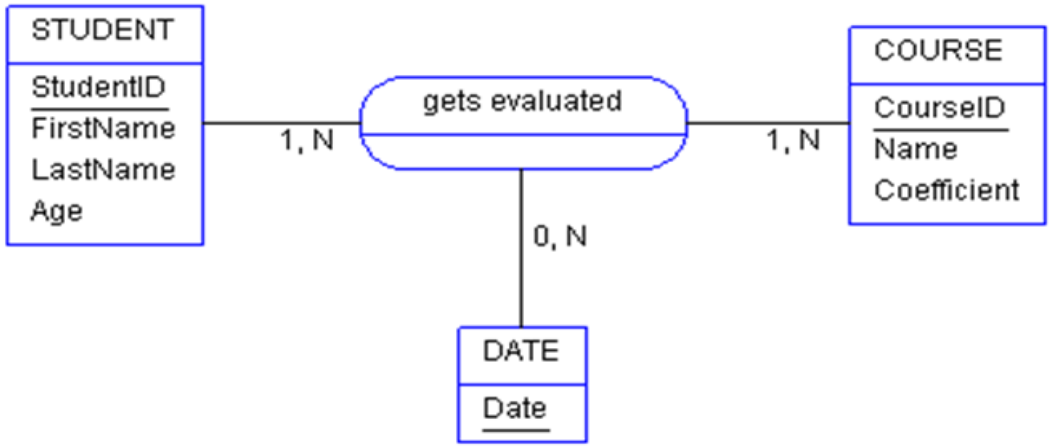
2.

Yes, a student can have multiple grades in the same course, recorded on different dates.
Justification: The primary key of the EVALUATION table (relation) is a composite key consisting of three attributes: StudentID, CourseID, and Date. Including the Date in the primary key ensures uniqueness, allowing multiple evaluations for the same student in the same course but on different dates.

3.

Attribute	Constraint	Observation
StudentID+CourseID+Date	Primary key: Unique and not null	The values must exist for all three attributes and must not be repeated for each value of the primary key.
StudentID CourseID Date	Domain Domain Domain	Not specified Not specified Date
StudentID CourseID	Referential Integrity Constraint (RIC) Referential Integrity Constraint (RIC)	EVALUATE.StudentID references STUDENT.StudentID EVALUATE.CourseID references COURSE.CourseID

4.

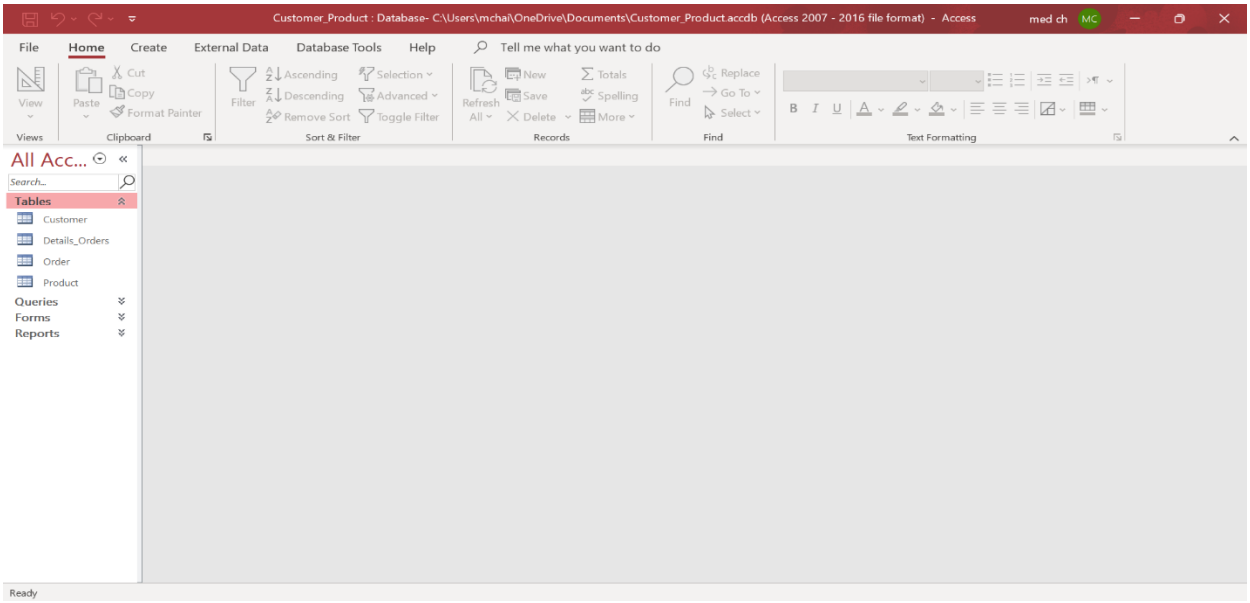


Chapter 5: Example of a DBMS (Database Management System) "Access"

1. Access interface

Microsoft Access provides a graphical interface that makes database management easier without necessarily writing SQL manually. It utilizes the same features found in all Office suite applications.

Main Interface: Ribbon menu with tabs for managing tables, forms, reports, and queries, as shown in the figure below.



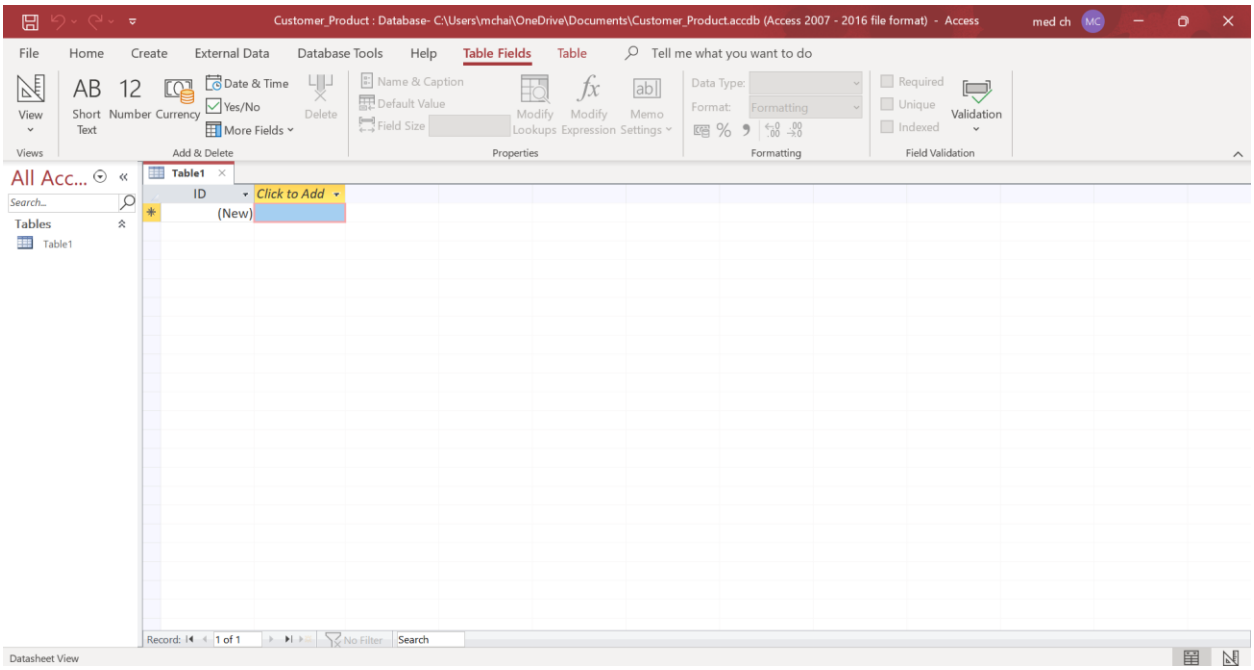
- **Tables**: Store structured data.
- **Queries**: Retrieve and modify data.
- **Forms**: User-friendly interface for data interaction.
- **Reports**: Generate formatted outputs for printing (e.g., invoices, client lists, etc).
- **Macros:** automate repetitive tasks by recording a sequence of actions (e.g., opening forms, running queries, or exporting data).

Presentation of the Working Database:

The working database is named ” Customer_Product”. This database contains 4 tables, including one association table (Details_Orders).

2. How to Create an Access Database?

1. Click on Start (Windows menu).
2. Search for the Microsoft Access (e.g., 2016) icon in your programs list.
3. Microsoft Office Access will open.
4. Click the Blank Database icon.
5. Enter the database name in the File Name (e.g., Customer_Product) field.
6. Click Create.
7. Click the file, then save as to choose where to save the database.
8. Access will automatically create a default table named Table1. Close this default table by clicking the Close button (X), as we will soon learn how to create a custom table.
9. If you open the folder you selected during setup, you will find the database saved.



3. Tables

A table is a structured database object that stores data in rows (records) and columns (fields), acting as the core repository for all information. It is designed by defining its schema (field properties and data types) before data entry, enabling efficient organization and retrieval for queries, forms, and reports.

3.1 Data Dictionary (Customer_Product example)

The following data dictionary describes the tables, their columns, and the associated data types. This data dictionary is used in this course to create all the tables of our database example.

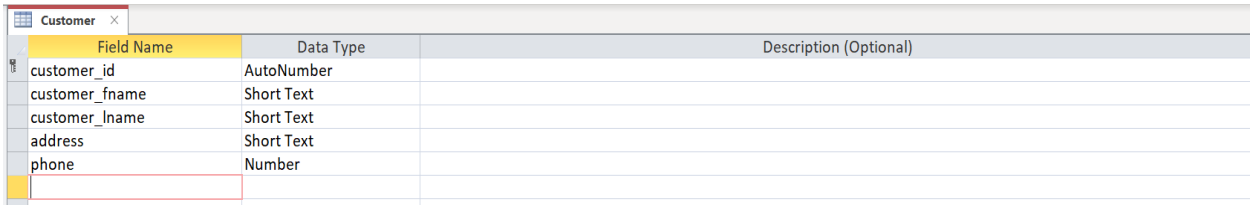
Example:

Table name	Column	Data type	Description
Customer	customer_id	AutoNumber	Primary key, unique customer identifier
	customer_fname	Text (20)	Customer's first name
	customer_lname	Text (20)	Customer's last name
	address	Text (100)	Full address of the customer
	phone	Number (10)	Customer's phone number
Product	producti_id	AutoNumber	Primary key, unique order identifier
	product_name	Text (20)	Name of the product
	unit_price	Currency	Unit price of the product
	stock	Number	Quantity of a product in stock
Order	order_id	AutoNumber	Primary key, unique order identifier
	order_date	Date/Time	Date of the order
	shipping_addr	Text (100)	Address where the order should be shipped
	customer_id	Number	Foreign key, references Customer table
Details_Orders	order_id	Number	Foreign key, references Order table
	product_id	Number	Foreign key, references Product table
	quantity	Number	Quantity of products in the order

3.2 Presentation of tables

3.2.1 Creating the structure of the Customer table using the data dictionary

1. To create our first table, we use the Design View. This mode is used to configure the structure of a table: defining the name and type of each field, and defining the primary key.
2. Using the data dictionary, fill in the "Field Name" and "Data Type" columns. Also, specify the "Field Size" property when necessary (in the "Field Properties" window).
3. Define the primary key of the table by positioning the cursor anywhere on the line of the appropriate field, then clicking the icon in the toolbar (or via right-click). The table in Design View appears as follows:


Customer

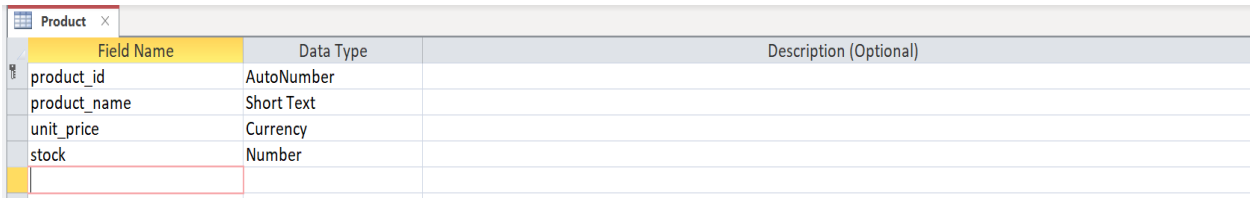
Field Name	Data Type	Description (Optional)
customer_id	AutoNumber	
customer_fname	Short Text	
customer_lname	Short Text	
address	Short Text	
phone	Number	

4. Save the table by clicking on the standard save icon.
5. You can now close the table by clicking the close button at the top right of the window.

3.2.2 Creating the structure of the 3 tables Product, Order, and Details_Orders

By following the same steps, the tables in Design View appear as follows:

1. Product table


Product

Field Name	Data Type	Description (Optional)
product_id	AutoNumber	
product_name	Short Text	
unit_price	Currency	
stock	Number	

Table = set of records (rows);
Record = set of fields (columns);
Column headers: names of the fields;
Each field has a specific data type (number, text, date, etc.). This type was chosen during the design of the table.

2. Order table

Order

Field Name	Data Type	Description (Optional)
order_id	AutoNumber	
order_date	Date/Time	
shipping_addr	Short Text	
customer_id	Number	

3. Details_Orders table

Details_Orders

Field Name	Data Type	Description (Optional)
order_id	Number	
product_id	Number	
quantity	Number	

3.2.3 Display the contents of an Access table (datasheet view).

- Double-click on its name in the navigation pane
- A window appears (similar to an Excel sheet).

P.S.: You can edit the data directly, but you should not do it now. Changes are saved as soon as you move to a different active cell.

1. Customer tables with values

Customer

customer_id	customer_fname	customer_lname	address	phone	Click to Add
1	Romaissa	Bourassi	45, avenue el moudjahidine	0789875324	
2	Riyad	Saidi	Algiers center	0678231098	
3	Lidya	Simoussi	100 Liberty Street, Kolea	0576234567	
4	Bougrida	Siham	10, 1st of May Square, Kolea	0795347689	

2. Product table with values

Product

product_id	product_name	unit_price	stock	Click to Add
1	Laptop	90,000.00 DA	50	
2	Smartphone	49,000.00 DA	150	
3	Laser Printer	20,000.00 DA	80	

3. Order table with values

Order

order_id	order_date	shipping_addr	customer_id	Click to Add
1	8/6/2024	45, avenue el moudjahidin	1	
2	3/27/2024	Algiers center	2	
3	9/16/2024	25 Independence Street, Cheraga	1	
4	4/14/2024	100 Liberty Street, Kolea	3	

4. Details_Orders table with values

Details_Orders

order_id	product_id	quantity	Click to Add
1	1	1	
1	3	2	
2	2	1	
3	1	1	

4 Relationships between tables

Until now, we have treated tables as separate entities. In reality, however, tables are interconnected (and this is frequently the case in practice).

4.1 Objective

Relationships avoid redundant data and manage what is called referential integrity.
The Customer_Product Database manages:

- The list of the company's customers;
- A list of products ;
- The orders placed;
- The details of each order (Order number, list of ordered products).

- The Customer table does not need the other tables (all the fields belong to this table only);

- The Product table as well;

- For an order, we need to know the customer number (customer_id). An order is identified by an automatic number (order_id);

- Since an order contains a list of products with the quantity ordered, these details are stored in the Details_Orders table.

- We then need to know the order number and the reference of the product (product_id), and this is also where the ordered quantity is entered;
- The two fields of the primary key are foreign keys.

Customer → Order: A customer can place multiple orders (one-to-many relationship).

- **Foreign Key**: customer_id in the Order table references customer_id in the Customer table.

Order → Details_Orders: An order can contain multiple products (one-to-many relationship).

- **Foreign Key**: order_id in the Details_Orders table references order_id in the Order table.

Product → Details_Orders: A product can be included in multiple orders (one-to-many relationship).

- **Foreign Key**: product_id in the Details_Orders table references product_id in the Products table.

4.2 Referential Integrity

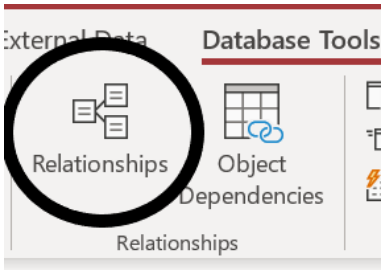
This prevents us, for example, from entering in the Orders table a customer_id that does not exist, or in the Details_Orders table an order_id or a product_id that does not exist.

Important note: Even though the customer_id fields in the Customer and Order tables have the same name, Access does not know they refer to the same information. It is the relationship that indicates this.

4.3 Create relationships between tables

The steps to create relationships between tables are as follows:

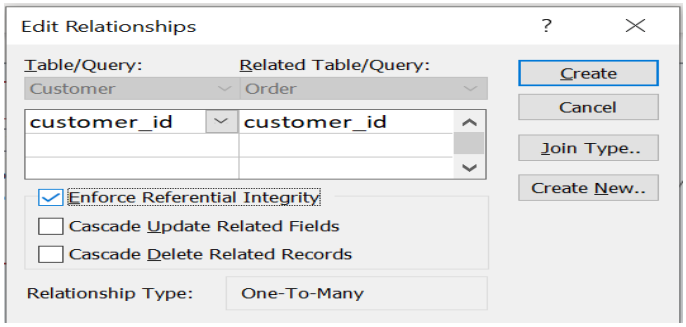
1. Close All Tables: Ensure all tables are closed; tables must be closed in order to create relationships between them.
2. Go to Relationships Tool: Click on the Database Tools tab and select Relationships.



The Database Tools tab manages relationships.

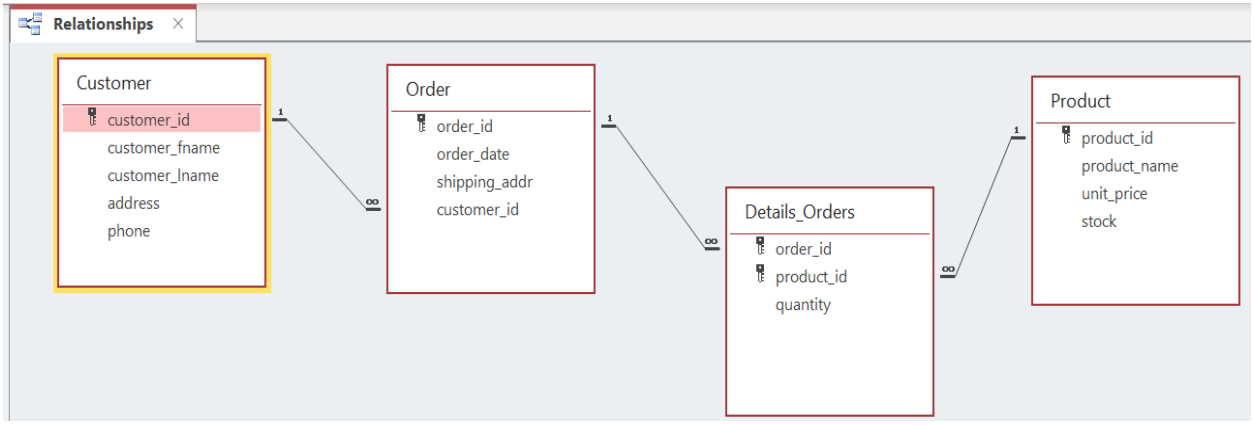
3. Add Tables: In the Relationships window, click Show Table, then select and add the tables (e.g., Customer, Order, Product, Details_Orders).
4. Create Relationships:
    - Drag customer_id from the Customer table to customer_id in the Order table.
5. Enforce Referential Integrity: Check Enforce Referential Integrity and click Create.

We have just created a 1-many relationship (each customer can have multiple orders).



6. **Save and Close**: Save your relationships and close the window.
7. Repeat the above step to create relationships between the tables **Order, Details_Orders**, and **Product**.

After creating the relationships between the tables, the final schema will look like this:



Tip: "1" in the relationship is always on the side containing the original key.

The database is now ready to be used (for example, queries on multiple tables).

4.4 Modify/Delete a Relationship

- Right-click on the relationship line;
- Choose the option to Modify or Delete.

5. Forms

5.1 What is a form?

A form is a user interface, a graphical interface that allows data from a table to be displayed or new data to be added to a table. Using a form, you can fill in a table.

A form can also be used to create menus, dialog boxes, etc.

You may have noticed earlier that we entered data directly into an open table (in datasheet view), which is not very user-friendly because it feels like entering data into Excel. However, using a form, we can input data in a more organized way to fill the table.

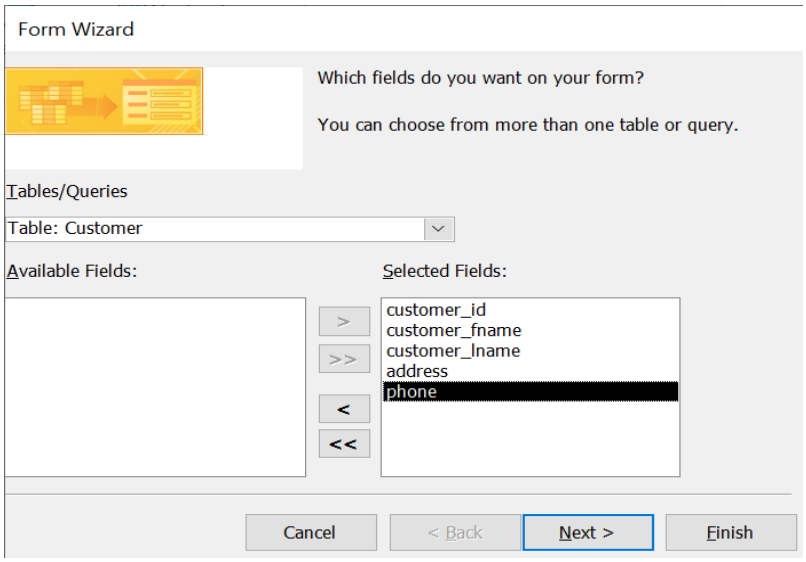
To use a form based on a specific table, we create a form from that table.

5.2 Creating Simple Forms

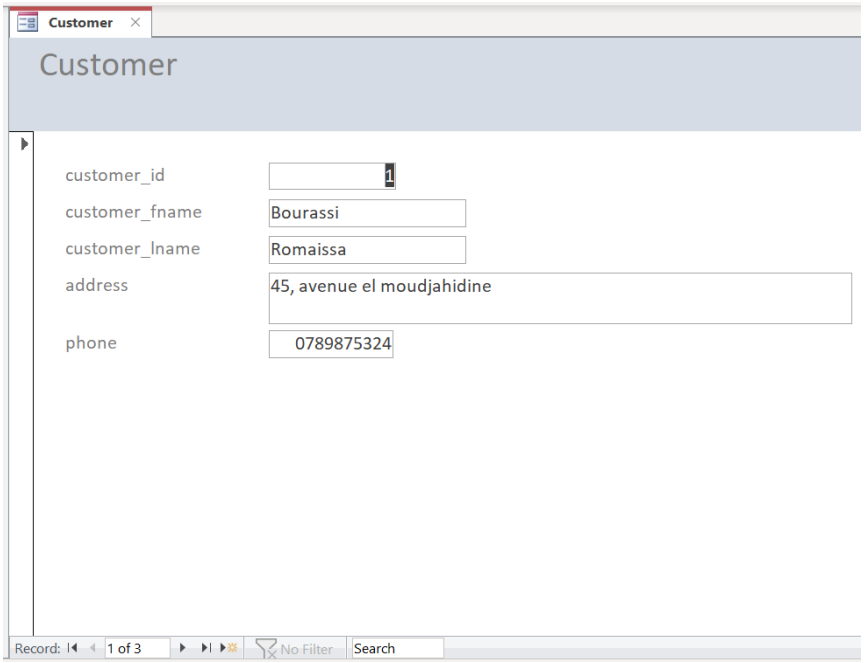
They are often linked to a single table or a single query and display their data.

5.2.1 Customer List (form linked to the Customers table)

- Open the "Details_Orders" database;
- Go to create tab -> Forms -> More Forms -> Form Wizard;
- In the window that appears, choose "Table: Customer";
- Move all the fields into the "Selected Fields" area by clicking the >> button;



- Next;
- Choose "Columnar" as the layout, then click Next;
- Enter the title of the form (name it "Customer") then click Finish.
- The form is created and displayed in "Form View," allowing you to browse and edit records.



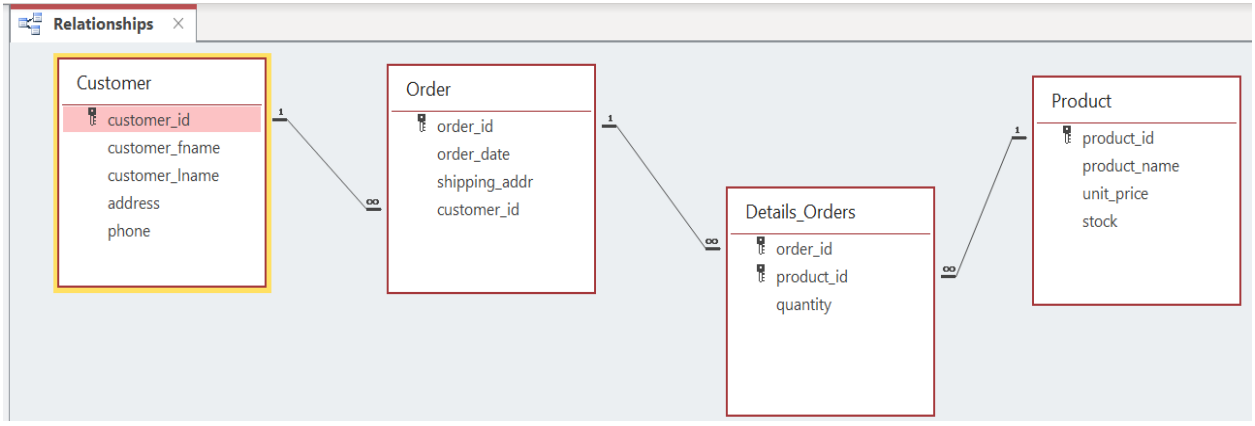
5.3 Creating advanced forms

In this section, we will focus on the following points:

- Using relationships between tables within forms;
- Adding/removing fields in a form;
- Adding controls;
- Forms based on queries.

5.3.1 Forms and related tables

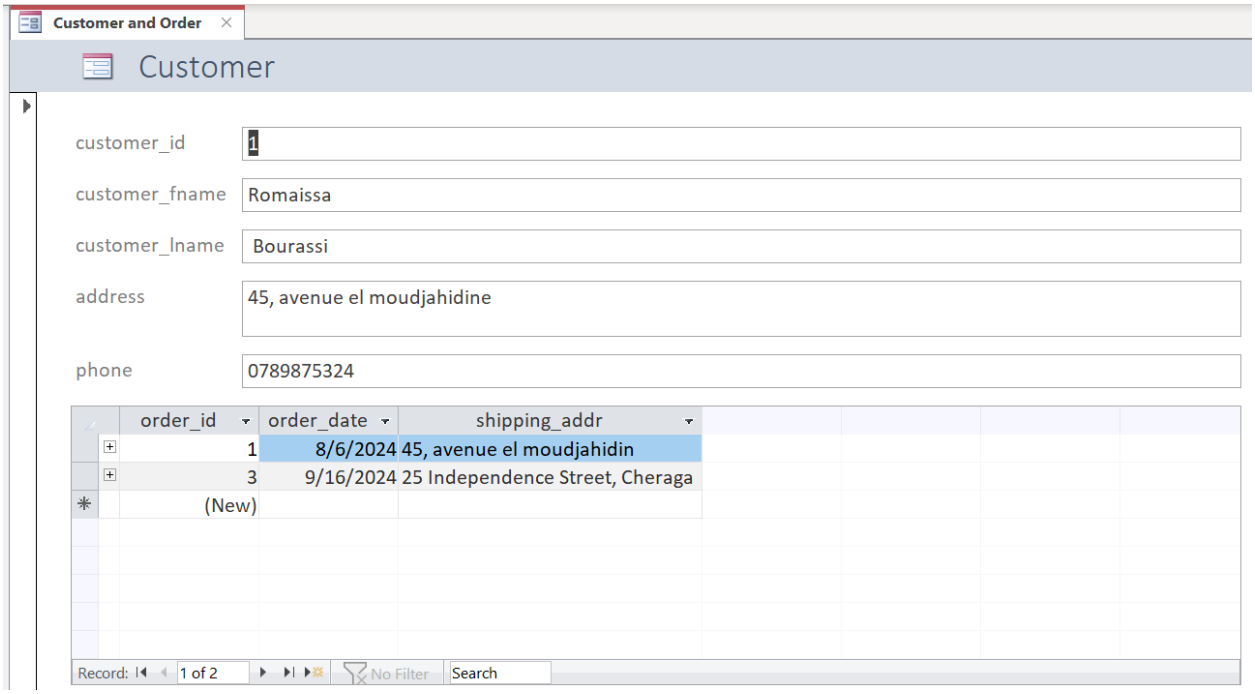
Here are the relationships that exist between the tables in the Facturation database:



Example: Form based on Customer and Order tables:

- In the Access navigation pane, select the Customer table;
- Go to Create tab -> Forms;
- Access then creates a form containing two sections:
  - The upper section displays the fields of the current Customer record;
  - The lower section contains (in the form of a datasheet) the orders of the current client.

- The figure below shows the obtained result.



Explanation: The Customer table is the parent table, and the Order table is the child table. The current record from the parent table is displayed at the top, and the corresponding records from the child table are displayed in the lower part in the form of a datasheet.

Advantage: The usefulness of such a layout is clear, for example, it allows the viewing or editing the orders of a single client.

Note: If a table has several child tables, Access uses the first table it encounters in the relationships.

6. Reports

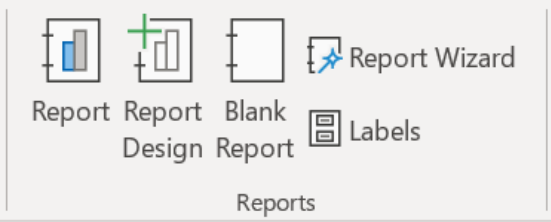
What is a Report?

A Report is the printout page for Access data. In Access, when we want to print something, we use the Report object.

Let's say we want to print a list of all customers. We must first create a Report corresponding to that list.

How to create a report?

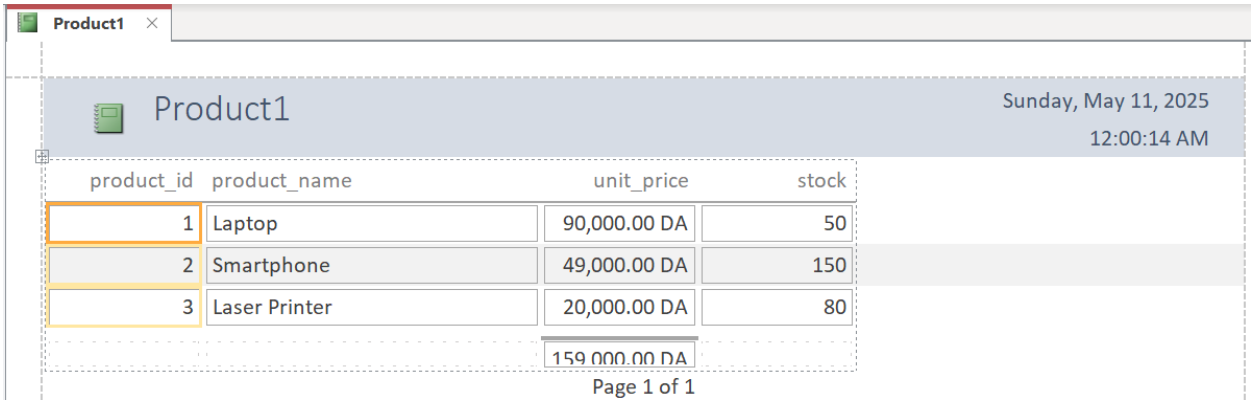
The Create tab -> Reports allows you to create the different types of reports:



Basic reports

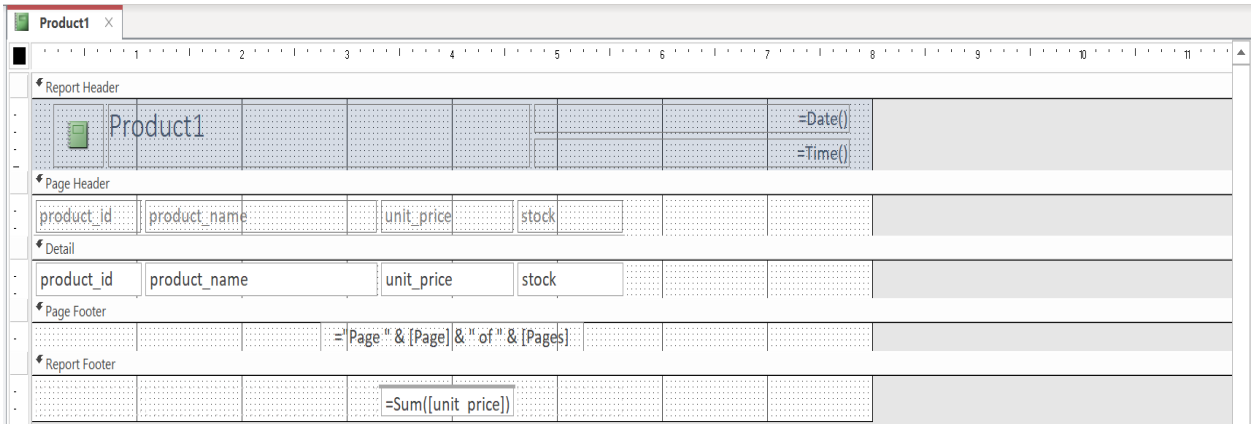
Create a simple report

- Click on the Product table in the Navigation Pane;
- Go to Create -> Reports -> Report;
- Access then creates a report displaying the contents of the catalogue (see the figure below);
- It is displayed in Print Preview mode;
- Save the report and name it "Product1".



The different display modes of a report

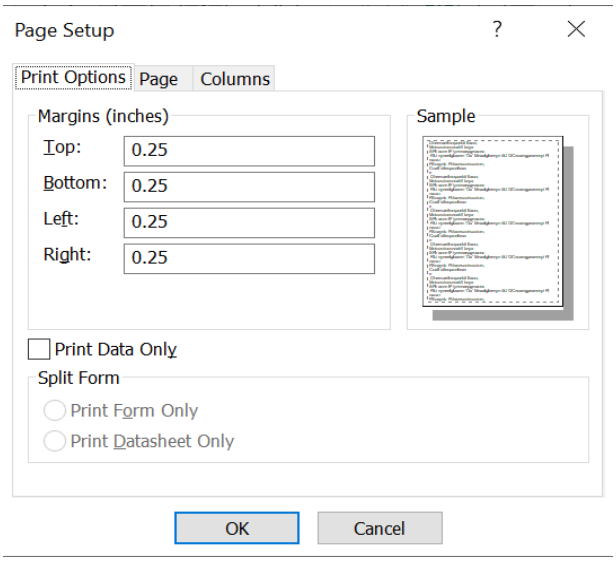
1. Design Mode:



- Allows more precise management of the various controls;
- A report contains the following sections:
  - A report header**;**
  - A page header (often contains column titles, such as table fields);
  - The Detail section contains the fields themselves;
  - The page footer (what will be displayed at the bottom of each printed page);
  - The report footer (what will be displayed after the last line on the last page, such as a total here).

2. Print Preview mode:

- Right-click on the title bar of the report -> Print Preview mode;
- The report is displayed as it will be printed;
- Right-click -> Page Setup;



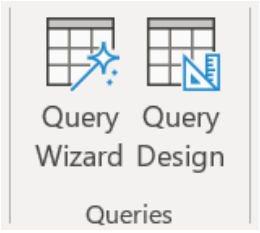
- You can then modify:
  - the margins,
  - the print orientation (Portrait, Landscape),
  - the paper size,
  - the number of columns (one or several tables side by side),
  - the spacing between lines,
  - etc

7. Queries

Queries are used to manipulate the data stored in tables. That is, in Access, tables are used only to store information. If you want to perform sorting, filtering, or calculations based on the data in the table, you use a query.

7.1 Types of Queries

- Select Queries (no data modification)
- Action Queries (or data modification queries)



The "Queries" group in the "Create" tab allows you to create queries.

7.2 Query features

- Combine multiple tables (A list of customers who ordered laptops last month: this information is found in different tables.);
- Perform calculations (e.g., sales revenue of a certain product);
- Create summaries through grouped data (e.g., grouping orders by customer to identify your main customer);
- Update data (deletion, modification, etc.) = Action Queries.

7.3 Query creation modes:

- Query Wizard (a quick but limited method);
- Design View (the most commonly used, based on a graphical interface);
- SQL View (requires writing formulas using SQL language, for advanced users).

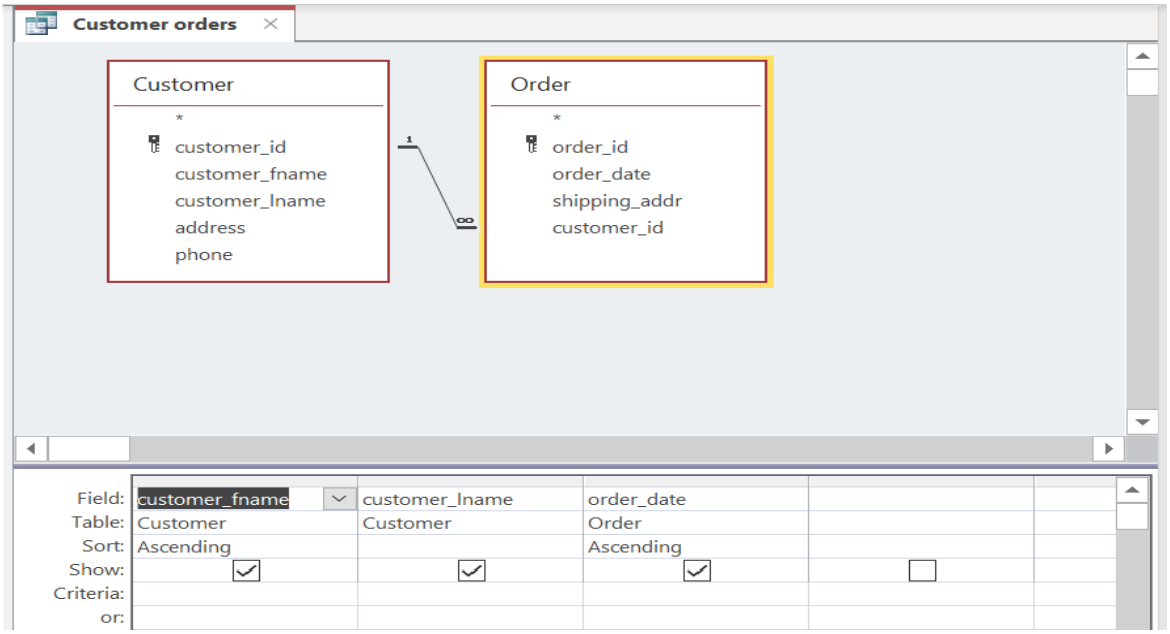
7.3 Select Queries

7.3.1 Query 1: In Design View – "Customer Orders"

Display the customers (first name and last name) along with their respective orders (Order Date). We want to sort the customers in alphabetical order and, for the same customer, sort the orders in chronological order.

- Close all windows;
- In the "Queries" group, click the “Query Design” button;
- In the window that appears, click on the Customer table, then click Add;
- Click on the Order table, then click Add;
- Notice that the relationship between the two tables is displayed;
- Double-click the "fname" field;
- Double-click the "lname" field;
- Double-click the "Order Date" field;
- For sorting, specify "Ascending" for both fields (the sorting is done based on the first column, then the second, and so on). To change the sort order, change the column order.
- Click the button 

in the Query Design -> Results;
- The result then appears in a datasheet;
- To switch to Design View, click on View -> Design View;
- If you don’t want a column to appear in the result, simply uncheck its "Show" box;
- Save the query as -> Customer Orders.



Explanation: It is the "Customer_id" field that allows Access to find a specific customer's orders. (It performs a cartesian product = join of the two tables, and keeps only the records that have the same Customer_id.).

7.3.2 Using Criteria in Queries

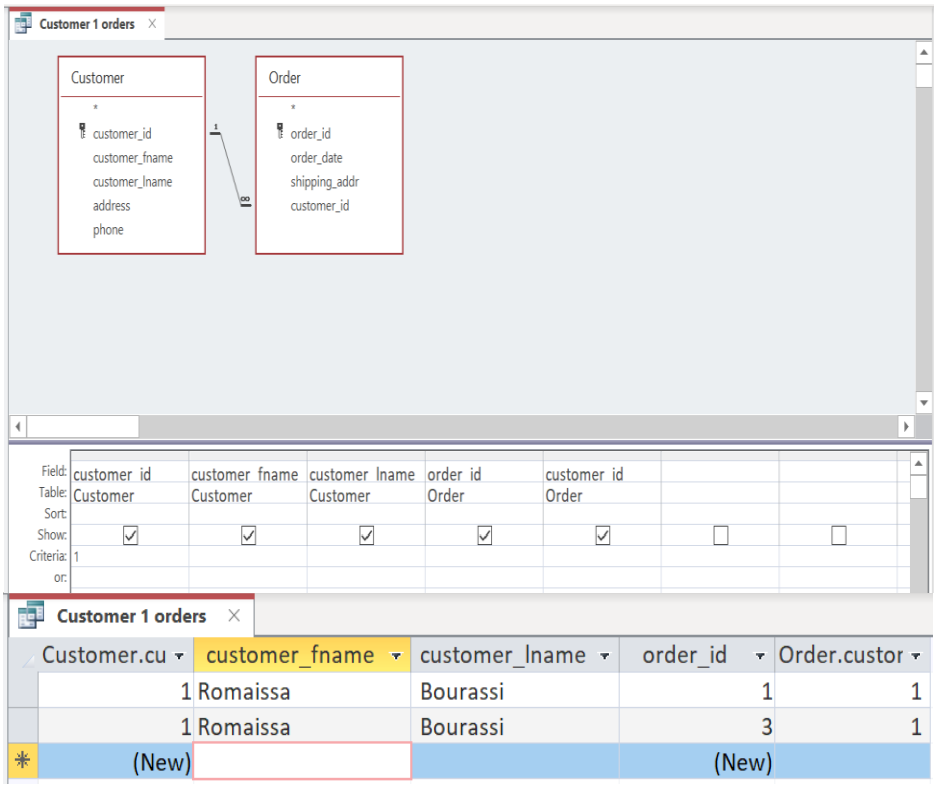
Criteria are used to filter the results of a query. A filter can be applied to fields of different types (numeric, text, date, etc.)

7.3.2.1 Criteria on Numeric Fields

You can use the operators **>**, **<**, **>=**, **<=**, **<>**, **AND**, **OR**, **NOT**.

Example: Orders from Customer 1 (Customer_id, first name, last name, order date)

Name the query: "Customer 1 orders".



	custom_id	
Criteria	1	Customer 1 or 4 (orders from both customers).
Or	3	For equality, you can use the **"="** sign, so you enter =1 and =3.

	custom_id	
Criteria	>=1 AND <=10	Customer 1 to 10 OR customer 12 (orders from these customers). AND is a logical operator.
Or	12	Two conditions, same column but on 2 different rows => Logical OR

	custom_id	
Criteria	1 OR 10	Customer 1 or 10 (orders from these customers) You can write: =1 OR =10
Or		

	custom_id	
Criteria	<>1	Orders from all customers except those of Customer_id 1
Or		

	custom_id	
Criteria	NOT 1	Orders from all customers except those of customer_id 1
Or		

	custom_id	order_id	
Criteria	1	Between 40 and 100	Orders from customer 1 where the order_id is >=40 and <=100. Both conditions must be verified when they are on the same row.
Or			

7.3.2.2 Criteria on Text Fields

The "**=**" is replaced by "**Like**"; Access often inserts it automatically, but if not, you need to enter it manually.

You can use the following wildcard characters:

- ***** replaces zero or more characters
- **?** replaces exactly one character
- **#** replaces exactly one digit

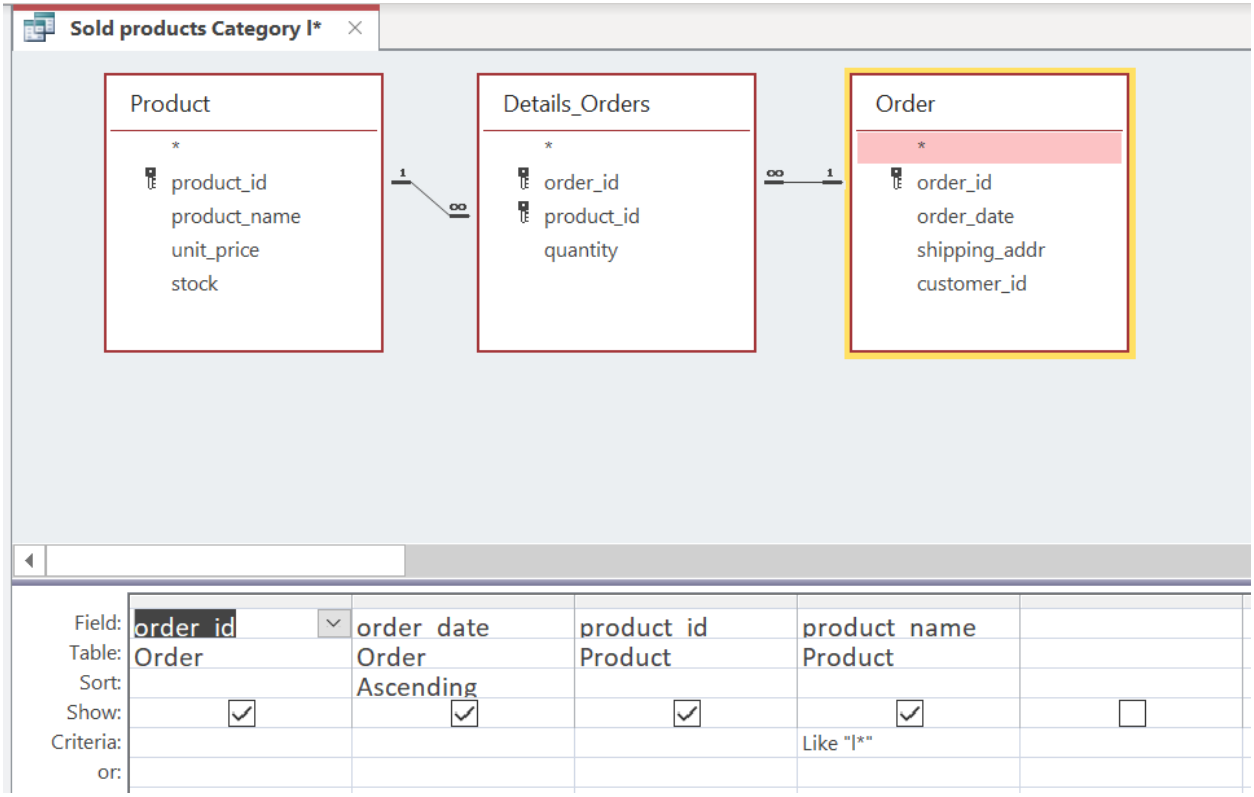
Examples of expressions using wildcards (no distinction between uppercase and lowercase):

- *Like "A*": text starting with the letter A
- Like "*ing": text ending with the string "ing"
- *Like "*c*m*": text containing 'c' followed by 'm', in that order
- Like "??1###": 2 characters + the digit 1 + any 3 digits

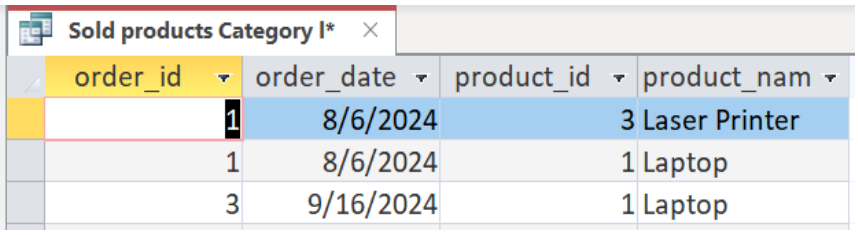
Example: We want to display the sold items whose category starts with the letter "a", and sort the orders in chronological order.

Display the fields: order id, order Date, product id, product name

Name the query: "Sold Products Category I*"



Result



7.3.2.3 Criteria on Date Fields:

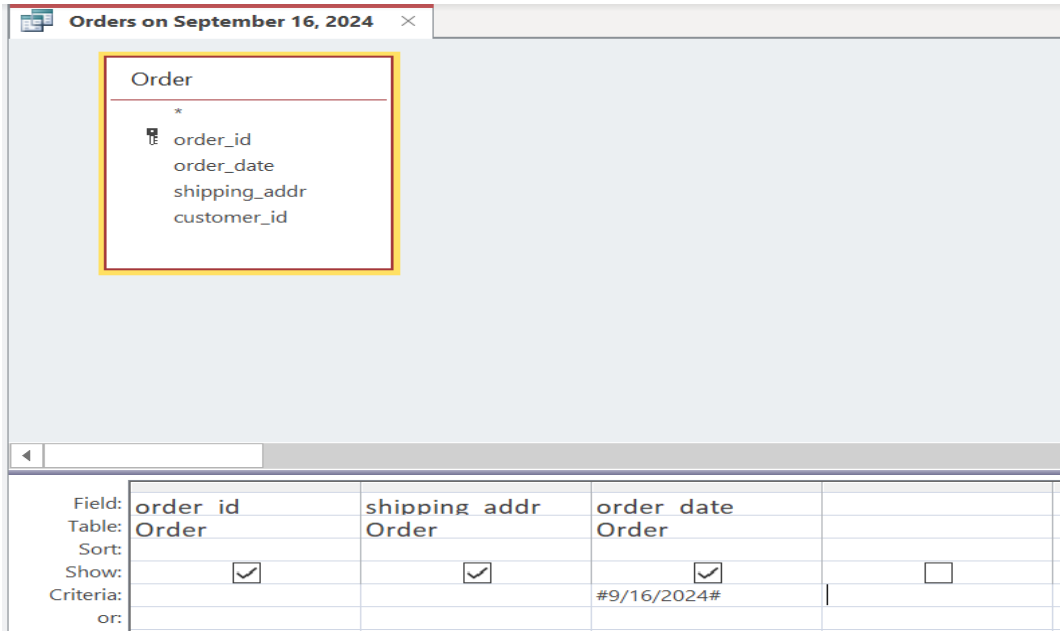
Similar to numeric types, the date is enclosed between two # symbols, e.g., #01/09/2008#.

You can use comparison operators such as <, <=, >, >= as well as date functions like Date().

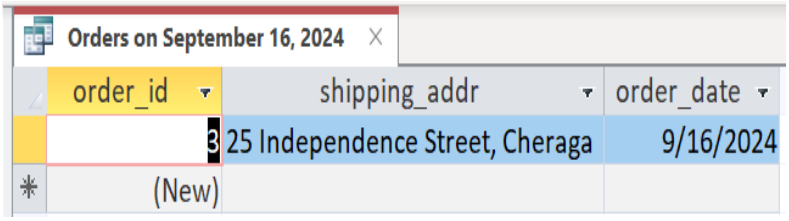
Example: Orders placed on September 16, 2024

Fields to display: order id, shipping address, order Date

Name the query: **"Orders on September 16, 2024"**



Result



Modify the criterion of the "order date" field

< #9/16/2024#: orders placed before this date

> #9/16/2024#: orders placed after this date

<= #9/16/2024#: orders placed on or before this date

>= #1/1/2024# AND <= #12/31/2024#: orders placed in 2024

Equivalent to: **Like "*/*/2024"**

< Date(): orders placed before today

> Date() - 30: orders placed within the last 30 days

< Date() - 30: orders placed more than 30 days ago

7.3.3 Parameterized Query

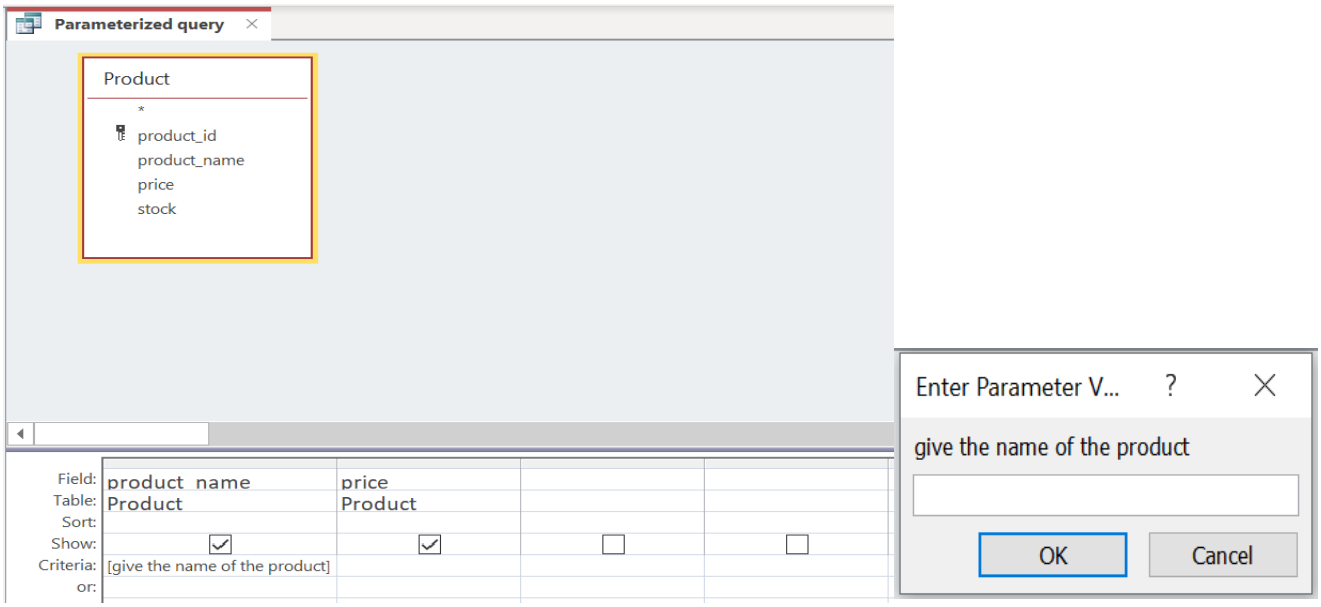
The criterion specified in a query for a field can be entered at the time the query is run.

To do this, simply place a message inside [ ] in the field's criteria.

Example: We want to display the price of a specific product, and we want to enter the product name at the time the query runs.

Fields to display: product name, unit price

Query name: Parameterized Query



- At execution, you will see the message "Enter the name of the product" in a dialog box containing an input field.
- Type the product name and click OK.
- The product price will then be displayed.

Note: You can use multiple parameters, there will be one dialog box for each parameter.

7.3.4 Creating Calculated Fields

Suppose we want to display the Unit Price including VAT (Value-Added Tax) in the result of a query.

This field does not exist in the tables, but it can be calculated from the Unit Price and VAT (for example, Unit Price including tax = Unit Price * 1.19), since the VAT field does not exist either.

The method consists of creating a calculated field in the query:

A field is identified by its name in square brackets. For example: [order_date] or [Order].[order_date]. The table name is only necessary if the query uses multiple tables and the same field appears in more than one table (to avoid ambiguity).

The square brackets are only used if the field name contains spaces.

Example: We want to display orders with their details and also calculate the Unit Price (Inc. VAT)

Display: Order_id, order_date, product_name, unit_price, Quantity, Amount (Excl. Tax), VAT, Amount (Incl. Tax)

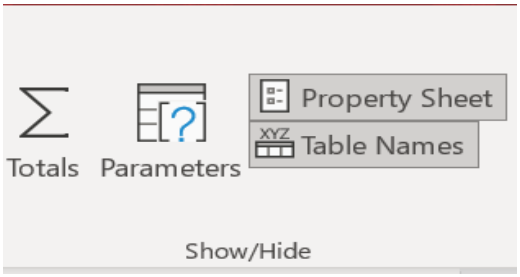
Sort: order_date in ascending order

Formulas: Amount (Excl. Tax): [unit_price] * [quantity], VAT: [Amount (Excl. Tax)] * 0.19, Amount (Incl. VAT): [Amount (Excl. Tax)] + [VAT]

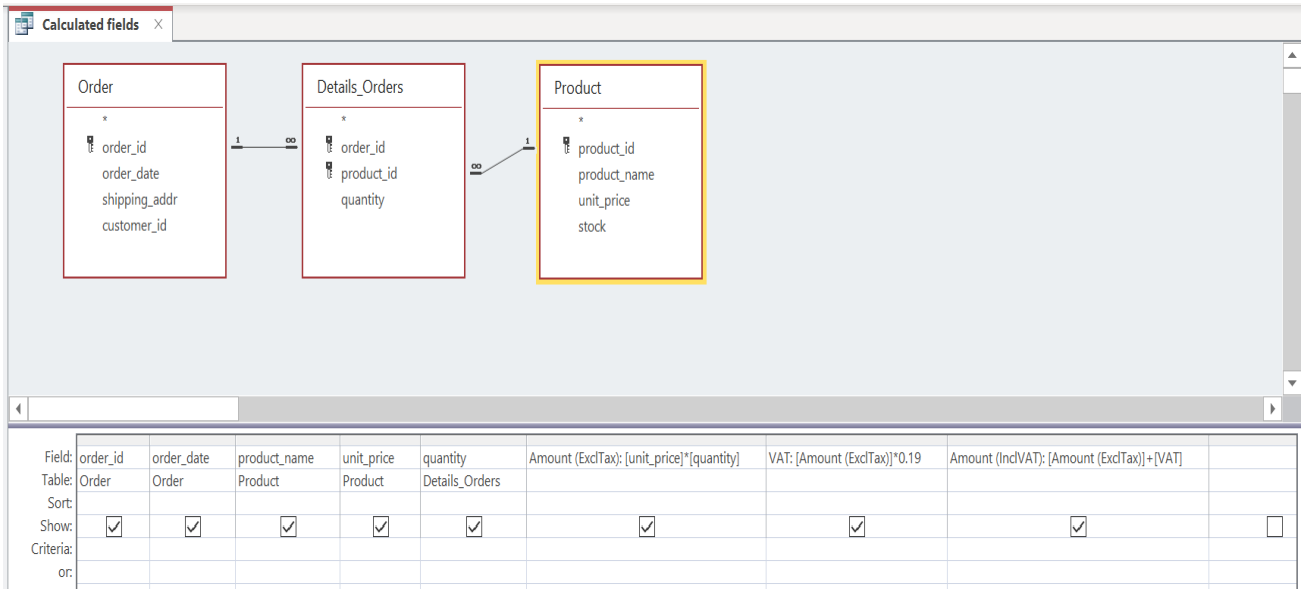
Query Name: Calculated Fields

Set the calculated fields to Currency format (those that are VAT)

- In the Query design tab, show/hide, click on Property sheet
- Format -> Currency.



Result



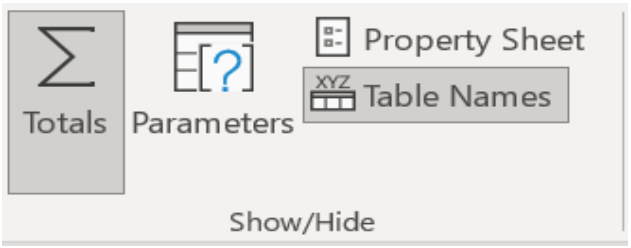
7.3.5 Summarizing Data

The queries we've seen so far consider records one by one.

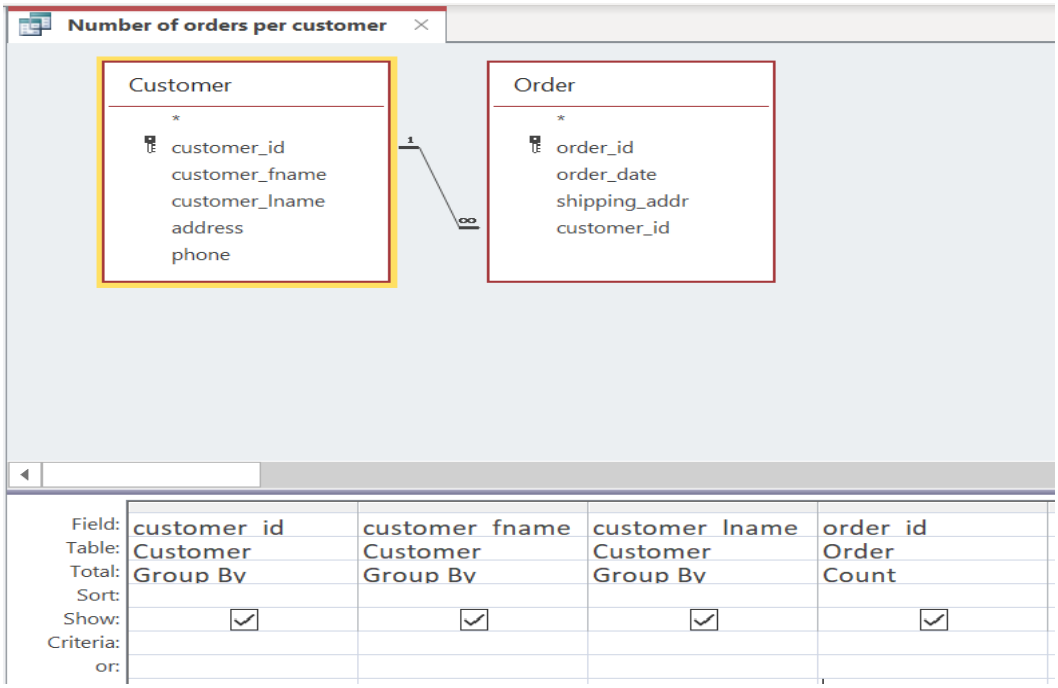
However, it is possible to group records in order to perform calculations (sum, average, etc.):

- Number of orders per customer
- Total amount paid by each customer
- Amount spent per product and per customer
- Etc.

To do this, use the "Totals" command in Query Design tab.

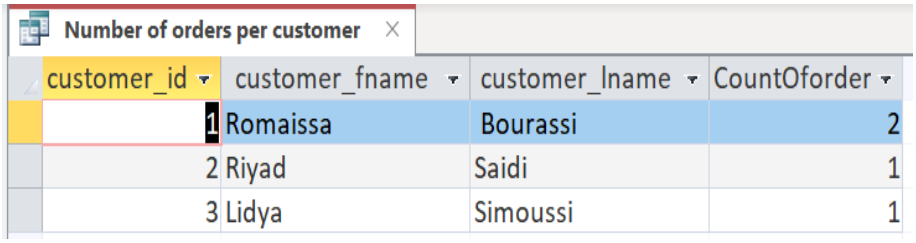


Example 1: Number of orders per customer



To display the Total row, simply click on the Totals button.

Result



We can check by displaying the contents of the Order table, sorted by customer_id.

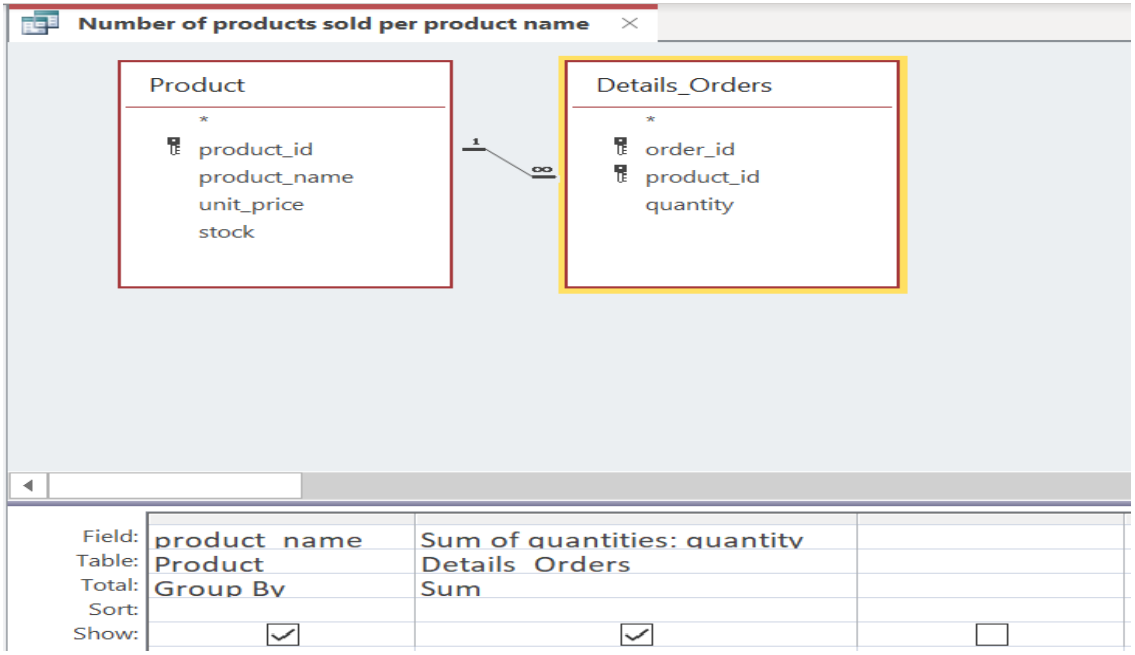
Name the query: Number of Orders per customer**.**

For each field, you can choose the operation to perform.

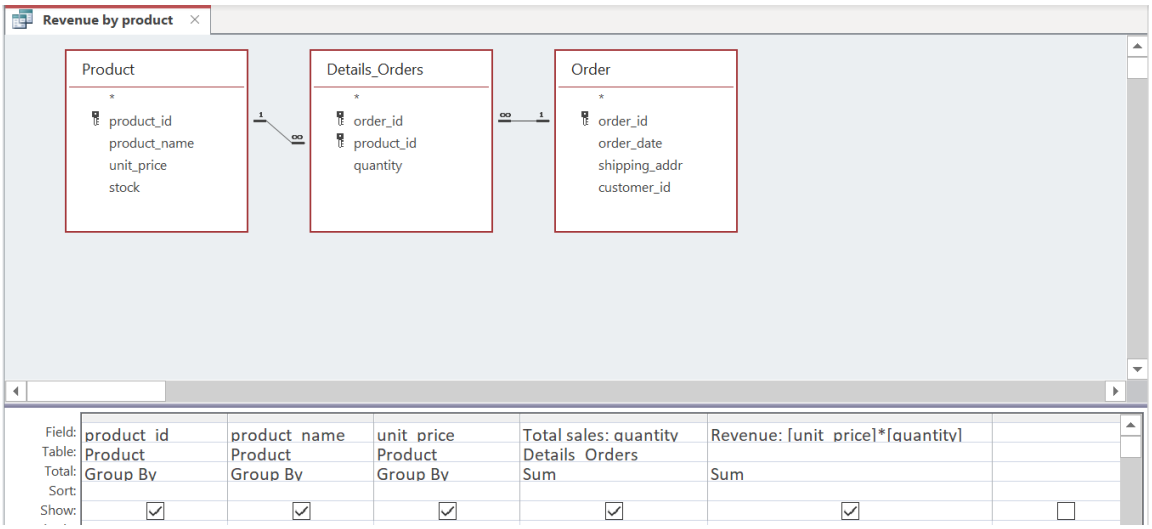
There are 3 types of operations:

- **Aggregation operations**: Sum, Average, etc.
- **Grouping operation**: The grouped values will be represented by a single row in the result.
- **Filtering operation**: the “Where” option.

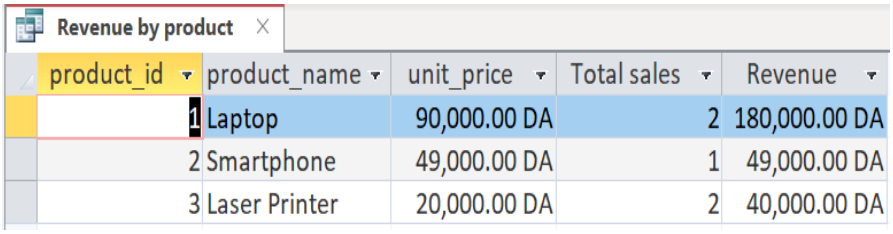
Example 2: Number of products sold per product name



Example 3: Revenue by product



Result



7.3.6 Other Query Creation Modes

Design View is the most commonly used method for creating queries, but you can also use:

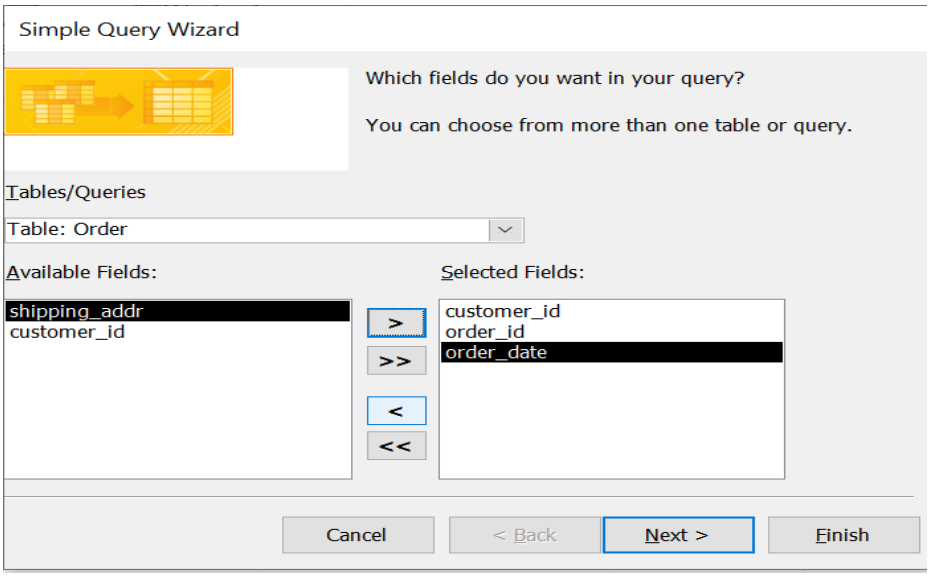
- the **Wizard** to create simple queries.
- the **SQL View** (*Structured Query Language*), which allows you to write queries directly using SQL.

7.3.6.1 Query Wizard

Example: Display all customer orders (customer id, order id, order_date)

- Create → Queries → Query Wizard
- Select the option "Simple Query Wizard"
- In the Table/Query list, choose the "Customer" table
- Select the field "customer_id", then click the **>** button
- Select the "Orders" table

- Choose the fields "Order_id" and "order_date"
- Click the "Next" button



- Choose the "Detail..." option
- Next
- Enter the name of the query: "Query Wizard"
- Finish
- The query is opened in "Datasheet View", and you can switch to "Design View" to edit it.

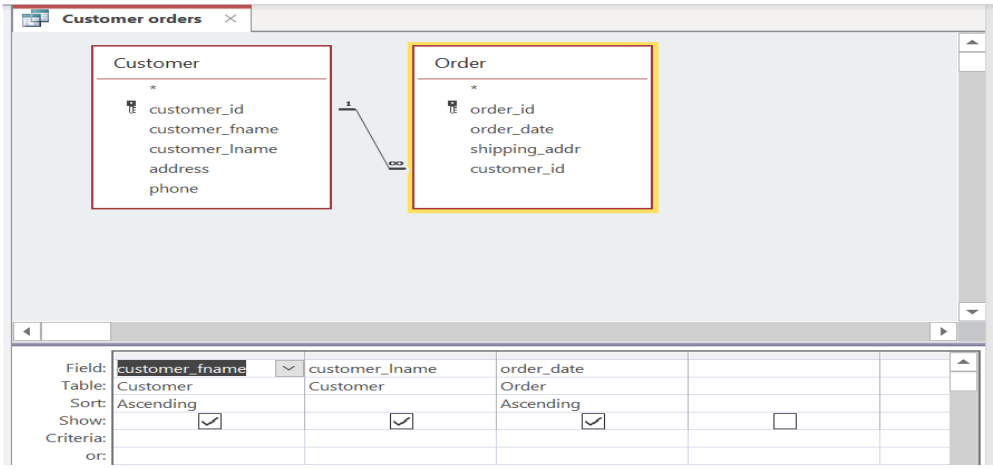
Conclusion: The "Query Design" mode is more practical and flexible.

7.3.6.2 SQL Mode

Primarily used by database experts;

In fact, Access generates SQL code every time a query is created.

Open the first query: Customer orders;



- Right-click on the title bar of the query window;
- Select SQL View;
- Here is the SQL code obtained:

SELECT Customer.customer_fname, Customer.customer_lname, Order.order_date
FROM Customer **INNER JOIN** [Order] **ON** Customer.customer_id = Order.customer_id
ORDER BY Customer.customer_fname, Order.order_date;

We will study this type of query in detail in the next chapter.

Chapter 6: SQL

6.1 SQL definition

SQL (Structured Query Language) is a standardized programming language specifically designed for managing and manipulating relational databases. SQL enables users to perform a wide range of operations, such as querying data, updating records, inserting new data, and managing database structures. SQL commands are divided into several categories, including:

1. **Data Query Language (DQL)**: Used for querying data (e.g., SELECT).
2. **Data Definition Language (DDL)**: Used for defining and managing database structures (e.g., CREATE, ALTER, DROP).
3. **Data Manipulation Language (DML)**: Used for inserting, updating, and deleting data (e.g., INSERT, UPDATE, DELETE).
4. **Data Control Language (DCL)**: Used for defining access controls and permissions (e.g., GRANT, REVOKE).

6.2 SQL Syntax and Structure

SQL syntax consists of specific keywords, operators, and punctuation that are used to write SQL statements. Key components include:

1. **Keywords**: Reserved words in SQL that perform specific functions, such as SELECT, FROM, WHERE, JOIN, etc.
2. **Identifiers**: Names of database objects, including tables, columns, and indexes.
3. **Clauses**: Components of SQL statements that specify conditions and filters (e.g., WHERE, ORDER BY).
4. **Operators**: Symbols used to perform operations on data, such as =, >, <, AND, OR, etc.

6.2.1 Data Definition Language:

1. Create a Table

To create a table, the keyword **CREATE TABLE** is used. The syntax is as follows:

```
CREATE TABLE TableName (
ColumnName1 DataType1 Constraint1,
ColumnName2 DataType2 Constraint2,
...
ColumnNameN DataTypeN ConstraintN
);
```

Column Constraints

- **PRIMARY KEY**: Declares the column as the primary key of the table.
- **NOT NULL**: Ensures that the column does not contain NULL values.
- **CHECK condition**: Ensures that all values in the column satisfy a given condition.
- **REFERENCES Tab(Col)**: Declares the column as a foreign key referencing column Col in table Tab.

Example:

```
CREATE TABLE Client (
ID INT PRIMARY KEY,
Name VARCHAR(30) NOT NULL,
Type VARCHAR(15) CHECK (Type IN ("Individual", "Company", "Other")),
City VARCHAR(20) NOT NULL,
Revenue FLOAT );
```

2. Delete a Table

The table is deleted using the instruction:

```
DROP TABLE TableName;
```

Example:

```
DROP TABLE Client;
```

6.2.2 Data Manipulation Language

- Insert a tuple into a table **(INSERT INTO)**

- Delete one or more tuples from a table **(DELETE FROM)**

- Update one or more tuples in a table **(UPDATE)**

1. INSERT INTO

INSERT INTO allows inserting one or more tuples into a table.

Syntax of the INSERT INTO instruction:

INSERT INTO TableName VALUES (v1, v2, ...);

Example:

```
INSERT INTO Client VALUES
    (1, "Morsli", "Individual", "Alger", 2000000),
    (2, "Saidi", "Company", "Blida", 3000000);
```

2. DELETE FROM

DELETE FROM deletes all tuples from the table that meet a condition.

Syntax of the DELETE FROM instruction:

```
DELETE FROM TableName
WHERE condition;
```

Example:

```
DELETE FROM Client
WHERE City = "Alger";
```

3. UPDATE

UPDATE performs the modification after the SET keyword on all tuples in the table that meet a condition.

Syntax of the UPDATE instruction:

```
UPDATE TableName
SET Column = Value
WHERE condition;
```

Example:

```
UPDATE Client
SET City = ''Blida''
WHERE Name = ''Morsli'';
```

6.2.3 Data Querying: SELECT

- The SELECT instruction is used to extract records from one or more tables.
- The SELECT instruction can be expressed in several ways: projection, restriction, join, etc.

The syntax for a query is represented as follows:

```
SELECT column1, column2, ..., columnN
FROM table1, table2, ..., tableM
WHERE condition1 AND condition2 ... AND conditionK;
```

1. Projection

Projection allows extracting one or more columns from a table.

The syntax for expressing projection:

```
SELECT column1, column2, ..., columnN
FROM tableM;
```

Example:

SELECT Name, City FROM Client;

Client table:

ID	Name	Type	City	Revenue
1	Morsli	Individual	Alger	2000000.00
2	Saidi	Company	Blida	5000000.00
3	Mosbahi	Company	Alger	1800000.00
4	Bougrida	Individual	Tipaza	3000000.00

Result:

Name	City
Morsli	Alger
Saidi	Blida
Mosbahi	Alger
Bougrida	Tipaza

Remark 1:

The * option allows extracting all columns from the table.

SELECT * FROM Client;

Result:

ID	Name	Type	City	Revenue
1	Morsli	Individual	Alger	2000000.00
2	Saidi	Company	Blida	5000000.00
3	Mosbahi	Company	Alger	1800000.00
4	Bougrida	Individual	Tipaza	3000000.00

Remark 2:

In SQL, projection does not eliminate duplicates in the result. To do this explicitly, use the keyword UNIQUE or DISTINCT.

Example: Return the cities of the clients.

SELECT City FROM Client;

Result:

City
Alger
Blida
Alger
Tipaza

SELECT DISTINCT City FROM Client;

Result:

City
Alger
Blida
Tipaza

2. Restriction

The restriction is expressed in the WHERE clause of the SELECT statement. It defines a condition that the tuples must meet.

The syntax for expressing a restriction:

SELECT column1, column2, ..., columnN

FROM table1...tableM

WHERE condition;

- The condition uses the following operators:
  - Comparison: >, =, <, >=, <=, <>
  - Boolean: NOT, AND, OR
  - Built-in: BETWEEN, IN, LIKE, IS NULL

Example 1:

Which clients are from the city of Algiers?

```
SELECT *
FROM Client
WHERE City = "Alger";
```

Result:

ID	Name	Type	City	Revenue
1	Morsli	Individual	Alger	2000000.00
3	Mosbahi	Company	Alger	1800000.00

Example 2:

Display the names of clients and their revenue in the city of Algiers, where the revenue is 2000000.00.

```
SELECT Name, Revenue
FROM Client
WHERE City = "Alger"
AND Revenue = 20000000;
```

Result:

Name	Revenue
Morsli	2000000.00

3. Other Operators

Examples of operators used:

1. **BETWEEN**: Tests if a value is within a range.
    - **Example:** Display clients whose revenue is between 2000000.00 and 5000000.00.

ID	Name	Type	City	Revenue
1	Morsli	Individual	Alger	2000000.00
2	Saidi	Company	Blida	5000000.00
3	Mosbahi	Company	Alger	1800000.00
4	Bougrida	Individual	Tipaza	3000000.00

```
SELECT Name, Revenue
FROM Client
WHERE Revenue BETWEEN 2000000 AND 5000000;
```

Result:

Name	Revenue
Morsli	2000000.00
Saidi	5000000.00
Bougrida	3000000.00

2. **IN**: Compares an expression to a list of values.

- Example: Display clients (Name) who live in Blida and Tipaza.

```
SELECT Name
FROM Client
WHERE City IN ("Blida", "Tipaza");
```

Result:

Name
Saidi
Bougrida

3. **LIKE**: Compares strings using a pattern.

- The symbol % replaces a string of any length.
- The symbol _ replaces a single character.

Example 1: Display clients whose names contain the letter "A".

```
SELECT Name, City
FROM Client
WHERE Name LIKE "%A%";
```

Result:

Name	City
Saidi	Blida
Mosbahi	Alger
Bougrida	Tipaza

Example 2: Display clients (Name, City) whose names have the letter 'O' in the second position.

```
SELECT Name, City
FROM Client
WHERE Name LIKE "_o%";
```

Result:

Name	City
Morsli	Alger
Mosbahi	Alger
Bougrida	Tipaza

4. ORDER BY

The ORDER BY clause allows sorting the result of a SELECT query in ascending (ASC) or descending (DESC) order.

Syntax:

```
ORDER BY column_1, column_2, ..., column_n ASC;
ORDER BY column_1, column_2, ..., column_n DESC;
```

- It sorts the results based on **column_1**.
- If two rows have the same values for **column_1**, they are sorted by **column_2**.
- And so on.

Example: Display all clients (Name, Revenue) in descending order of their revenue.

```
SELECT Name, Revenue
FROM Client
ORDER BY Revenue DESC;
```

Result:

Name	Revenue
Saidi	5000000.00
Bougrida	3000000.00
Morsli	2000000.00
Mosbahi	1800000.00

5. Cartesian Product

The **Cartesian product** of two relations **R1** and **R2** is a relation where:

- The columns consist of the columns from **R1** and the columns from **R2**.
- The rows are the combination of each row from **R1** with every row from **R2**.

R1

A	B
a	b
c	b
d	e

R2

B	C
b	c
e	a
b	d

R1 × R2 (Cartesian Product of R1 and R2)

A	R1.B	R2.B	C
a	b	b	c
a	b	e	a
a	b	b	d
c	b	b	c
c	b	e	a
c	b	b	d
d	e	b	c
d	e	e	a
d	e	b	d

Example: Display clients (ClientName, ClientCity) who placed orders during March 2022.

Order Table:

OrderID	OrderDate	Amount	ClientID
1	05/03/2022	50,000	1
2	11/03/2022	100,000	2
3	04/06/2022	128,000	3
4	24/09/2022	300,000	3

Client Table:

ClientID	ClientName	ClientCity
1	Morsli	Algiers
2	Saidi	Blida
3	Mosbahi	Algiers

```
SELECT ClientName, ClientCity
FROM Order, Client
WHERE OrderDate BETWEEN "03/01/2022" AND "03/31/2022";
```

Cartesian Product Result

OrderID	OrderDate	Amount	ClientID	ClientID	ClientName	ClientCity
1	05/03/2022	50,000	1	1	Morsli	Algiers
1	05/03/2022	50,000	1	2	Saidi	Blida
1	05/03/2022	50,000	1	3	Mosbahi	Algiers
2	11/03/2022	100,000	2	1	Morsli	Algiers
2	11/03/2022	100,000	2	2	Saidi	Blida
2	11/03/2022	100,000	2	3	Mosbahi	Algiers
3	04/06/2022	128,000	3	1	Morsli	Algiers
3	04/06/2022	128,000	3	2	Saidi	Blida
3	04/06/2022	128,000	3	3	Mosbahi	Algiers
4	24/09/2022	300,000	3	1	Morsli	Algiers
4	24/09/2022	300,000	3	2	Saidi	Blida
4	24/09/2022	300,000	3	3	Mosbahi	Algiers

6. Join

- The join of two relations (R1 and R2) is a relation R3, where the tuples are obtained by concatenating the tuples from R1 with those from R2.
- Only the tuples that satisfy the join condition are retained.
- The join expresses an equality between the primary key of one table and the corresponding foreign key of the other.

Example: Display clients who placed orders during March 2022.

Order Table:

OrderID	OrderDate	Amount	ClientID
1	05/03/2022	50,000	1
2	11/03/2022	100,000	2
3	04/06/2022	128,000	3
4	24/09/2022	300,000	3

Client Table:

ClientID	ClientName	ClientCity
1	Morsli	Algiers
2	Saidi	Blida
3	Mosbahi	Algiers

SELECT * FROM Order, Client

WHERE OrderDate BETWEEN "03/01/2022" AND "03/31/2022"

AND Order.ClientID = Client.ClientID;

Join Result

OrderID	OrderDate	Amount	ClientID	ClientID	ClientName	ClientCity
1	05/03/2022	50,000	1	1	Morsli	Algiers
2	11/03/2022	100,000	2	2	Saidi	Blida

7. Aggregation Functions

Function	Meaning
COUNT	Returns the number of rows in a given column
MAX	Returns the largest value in a given column
MIN	Returns the smallest value in a given column
AVG	Returns the average of the values in a given column
SUM	Returns the sum of the values in a given column

- It Can only be used on projection columns.

- It is possible to use several functions in a single SELECT clause.

Example: How many clients are there in Algiers?

ID	Client Name	Client Type	City	Revenue
1	Morsli	Individual	Algiers	2,000,000.00
2	Saidi	Company	Blida	5,000,000.00
3	Mosbahi	Company	Algiers	1,800,000.00
4	Bougrida	Individual	Tipaza	3,000,000.00

```
SELECT COUNT(ID) AS Nbr_Clients
FROM Client
WHERE City = ''Alger'';
```

Nbr_Clients
2

8. GROUP BY Clause

- By default, **COUNT**, **MIN**, **MAX**, **AVG**, and **SUM** are applied to all the selected tuples.
- It is possible to apply these functions to a partition of the results delivered by a query using the **GROUP BY** clause.

Syntax:

GROUP BY column_1, column_2, ..., column_n

- The **GROUP BY** instruction partitions the result of a query into subgroups.
- Each of these subgroups shares the same value for **column_1, column_2, ...column_n**.
- The functions are then applied to each of these subgroups.

Example: Provide the number of products for each category.

CodeProd	Designation	Category	Price	Stock Quantity
1	Lamp	Home	4000	20
2	Remote-controlled car	Toy	5000	50
3	Sports glove	Sport	1600	35
4	Sheet	Home	2600	40

```
SELECT Category, COUNT(CodeProd) AS Number_Prod
FROM Product
GROUP BY Category;
```

Result:

Category	Number_Prod
Home	2
Toy	1
Sport	1

9. HAVING Clause

The HAVING clause performs a selection (in the same way as WHERE) on the groups to be retained.

Example: Provide the number of products for each category where the total stock quantity is greater than 40.

CodeProd	Designation	Category	Price	Stock Quantity
1	Lamp	Home	4000	20
2	Remote-controlled car	Toy	5000	50
3	Sports glove	Sport	1600	35
4	Sheet	Home	2600	40

```
SELECT Category, COUNT(CodeProd)AS Number_Prod, SUM(Stock_Quantity AS Total_Stock_Quantity
FROM Product
GROUP BY Category
```

HAVING SUM(Stock_Quantity) > 40;

Result:

Category	Number_Prod	Total_Stock_Quantity
Home	2	60
Toy	1	50

10. Subquery

- Also called a nested query, it is a query within another query (the main query).
- It allows you to avoid executing two queries separately.

Example 1: Display clients who live in the same city as the client "Morsli."

ID	Client Name	Client Type	City	Revenue
1	Morsli	Individual	Algiers	2,000,000.00
2	Saidi	Company	Blida	5,000,000.00
3	Mosbahi	Company	Algiers	1,800,000.00
4	Bougrida	Individual	Tipaza	3,000,000.00
5	Achouri	Company	Algiers	7,000,000.00

```
SELECT *
FROM Client
WHERE City = (SELECT City
              FROM Client
              WHERE ClientName = "Morsli");
```

Result:

ID	Client Name	Client Type	City	Revenue
1	Morsli	Individual	Algiers	2,000,000.00
3	Mosbahi	Company	Algiers	1,800,000.00
5	Achouri	Company	Algiers	7,000,000.00

Explanation:
This query selects all clients who live in the same city as the client named "Morsli." The subquery retrieves the city of "Morsli" and uses that to find all other clients living in the same city.

In the case where multiple tuples can be returned, we can use:

- **IN**: to test membership in the set of what is returned.
- A comparison operator (<, >, ...) followed by ANY or ALL to test if the comparison is true at least once or for all tuples."

Example 2: Display the clients who live in the same city as the clients Morsli and Saidi.

ID	Client Name	Client Type	City	Revenue
1	Morsli	Individual	Algiers	2,000,000.00
2	Saidi	Company	Blida	5,000,000.00
3	Mosbahi	Company	Algiers	1,800,000.00
4	Bougrida	Individual	Tipaza	3,000,000.00
5	Achouri	Company	Algiers	7,000,000.00

```
SELECT *
FROM Client
WHERE City IN (SELECT City
                FROM Client
               WHERE ClientName = "Morsli"
               OR ClientName = "Saidi");
```

Result:

ID	ClientName	ClientType	ClientCity	Revenue
1	Morsli	Individual	Algiers	2,000,000
2	Saidi	Company	Blida	5,000,000
3	Mosbahi	Company	Algiers	1,800,000
5	Achouri	Company	Algiers	7,000,000

Example 3: Display products that are more expensive than those in the House category.

Product Table

ProductCode	Description	Category	Price	Stock_Quantity
1	Lamp	House	4000	20
2	Remote-controlled car	Toy	5000	50
3	Sports glove	Sport	1600	35
4	Sheet	House	2600	40

```
SELECT *
FROM Product
WHERE price > ALL (SELECT price FROM Product
                   WHERE Catégorie = "House");
```

Result:

ProductCode	Description	Category	Price	Stock_Quantity
2	Remote-controlled car	Toy	5000	50

Exercise

Given the following relational database:

CUSTOMER (CustomerID, FirstName, LastName, BirthDate, Address, Phone)

ORDER (OrderNo, Date, #CustomerID)

PRODUCT (Reference, Name, Price, Stock)

ORDER_PRODUCT (#OrderNo, #Reference, Quantity)

Questions

Write the following queries in SQL:

1. Give the reference and name of products with a price greater than 1000.
2. Give the first and last names of the customers who placed an order on October 23, 2022.
3. Give the number of products in each order.
4. Give the name and phone number of customers who ordered products with a quantity greater than 100.
5. Give the name of the products ordered by the customer 'Mohamed'.
6. For each customer, give the customer id, name (first name and last name), and total amount to be paid for all orders.
7. Give the clients who have more than two orders.
8. Give the products with the total quantity ordered since July 05, 2022, in cases where this quantity exceeds 60.
9. Give the products with price is above the average.
10. Give the product with the minimum price.

Correction

1. SELECT Reference, Name
   FROM PRODUCT
   WHERE Price > 1000;

2. SELECT Name
   FROM CUSTOMER, ORDER

```
WHERE CUSTOMER.CustomerID= ORDER.OrderNo
AND Date = ''10/23/2022'';
```

3.
```
SELECT OrderNo, COUNT(Reference)
FROM ORDER_PRODUCT
GROUP BY OrderNo;
```

4.
```
SELECT FirstName, LastName, Phone
FROM CUSTOMER, ORDER, ORDER_PRODUCT
WHERE CUSTOMER.CustomerID = ORDER.CustomerID
AND ORDER.OrderNo = ORDER_PRODUCT.OrderNo
AND Quantity > 100;
```

5.
```
SELECT Name
FROM PRODUCT, CUSTOMER, ORDER, ORDER_PRODUCT
WHERE CUSTOMER.CustomerID = ORDER.CustomerID
AND ORDER.OrderNo = ORDER_PRODUCT.OrderNo
AND PRODUCT.Reference = ORDER_PRODUCT.Reference
AND FirstName = ''Mohmaed'';
```

6.
```
SELECT CustomerID , FirstName, LastName, SUM (Price*Quantity) AS Amount
FROM CUSTOMER, ORDER, PRODUCT, ORDER_PRODUCT
WHERE CUSTOMER.CustomerID = ORDER.CustomerID
AND PRODUCT.Reference = ORDER_PRODUCT.Reference
AND ORDER.OrderNo = ORDER_PRODUCT.OrderNo
GROUP BY CustomerID, FirstName, LastName;
```

7.
```
SELECT CustomerID, Count(*)
FROM ORDER
GROUP BY CustomerID
HAVING Count(*) > 2;
```

8.
```
SELELCT Reference, SUM(Quantity)
FROM ORDER, ORDER_PRODUCT
WHERE ORDER.OrderNo = ORDER_PRODUCT.OrderNo
```

```
    AND Date >= "07/05/2022"
    GROUP BY Reference
    HAVING SUM(Quantity) > 60 ;
```

9.
```
SELECT *
FROM PRODUCT
WHERE Price > (SELECT AVG(Price)
                FROM PRODUCT);
```

10.
```
SELECT *
FROM PRODUCT
WHERE Price = (SELECT MIN(Price)
                FROM PRODUCT);
```

Supplementary exercises

Exercise 1

Here is the textual representation of a Logical Data Model (LDM) for an association that organizes car racing competitions. Each competition includes several races, all held on the same day.

Competition (compId, compName)
Race (raceId, startTime, distance, #compId)
Driver (driverId, driverName, phone)
Car (carId, brand, model, #driverId)
Result (#raceId, #driverId, rank, duration)

Rank: indicates the ranking of a driver in a race (positive integer).

Questions

1) Complete the table below with the missing information related to the following columns: Integrity constraint, type of constraint, and field(s) associated**.**

integrity constraint	type of constraint	field(s) associated
a. A constraint that ensures each car is associated with a driver	…… ……	…… ……
b. A constraint that guarantees …… …… …… ……	Domain Constraint	rank
c. A constraint that guarantees that a driver can have only one result per race.	…… ……	…… ……

3) Provide the steps to create the relationship between the two tables, Driver and Car.

4) Write the following queries in SQL:

1. Display all competitions.
2. List the drivers who finished a race in first place.
3. Display the cars with their drivers.
4. Calculate the average time of the cars for each race.
5. Display the drivers with an average time under 100, along with the total number of races each driver participated in.

Exercise 2

The zoo management database is represented in the following logical model:

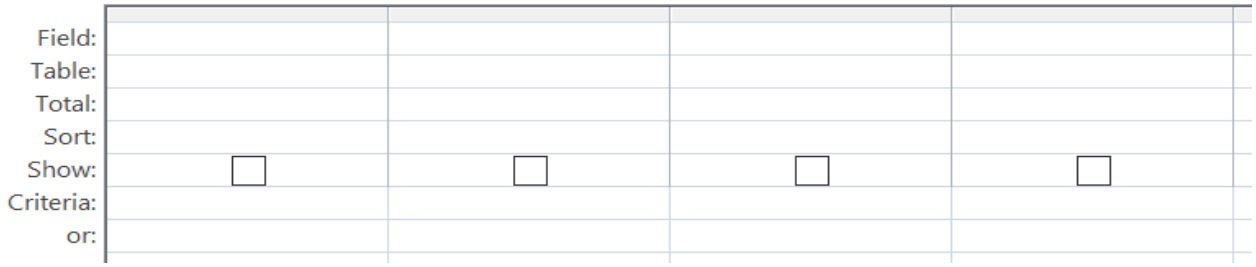
Zone (zoneCode, location)
Cage (cageNumber, area, #zoneCode)
Animal (animalID, animalName, #cageNumber, #familyID, countryOfOrigin)
Family (familyID, familyName)
VaccineHistory (#animalID, #vaccineID, vaccineDate)
Vaccine (vaccineID, vaccineName)

Questions

1. Complete the table below by placing a cross (X) in the True or False column.

Statement	True	False
a. A cage can be located in multiple zones.		
b. An animal belongs to multiple families.		
c. An animal can receive the same vaccine on different dates.		
d. An animal cannot receive multiple vaccinations		

2. Complete the table below to answer the following query:
   Display the number of animals in cages that contain at least 2 animals.



3. Write the following queries in SQL:

a. Display all animals and their country of origin.
b. List the countries of origin of the animals, without duplicates.
c. Display the names of animals whose family name starts with the letter "A".
d. Find the total number of animals per cage.
e. Display the number of animals in cages that contain at least 2 animals.

Exercise 3

Consider the simplified database described by the following relational schema:

CLIENT (ClientNumber, LastName, FirstName, Address)
SUPPLIER (SupplierCode, SupplierName)
CARD (CardNumber, Amount, SupplierCode)
LINE (LineNumber, ClientNumber, SupplierCode, Price, ActivationDate)

Questions

Part A:

1. List the keys of the above schema.

2. Complete the following statements.
a. The property that requires the LineNumber field not to be empty is ……
b. The property that allows adding the constraint (>0) to the Amount field is ……
c. The property that assigns a default value to the ActivationDate field is ……
d. The property that ensures each supplier is properly used as a card supplier is……..

3. Describe the procedure for creating the relationship between the two tables, CLIENT and LINE.

Part B:

1. Complete the table below to answer the following query:

a. What is the total number of lines activated by each client? Display only the clients who have activated at least 2 lines.

Field:				
Table:				
Total:				
Sort:				
Show:	☐	☐	☐	☐
Criteria:				
or:				

2. Write the following queries in SQL:

a. Display the line numbers that were activated after 01/01/2025.

b. Display the number of cards sold with an amount of 1000 DA.

c. Display the total sales amount of the cards belonging to the supplier named "ALGCOM".

d. What is the total number of lines activated by each client? Display only the clients who have activated at least 2 lines.

Exercise 4

Company X is organized as follows: it comprises employees, each uniquely identified by an employee number. Their name and gender are also recorded. Each employee must belong to one and only one professional category. Each category is defined by a category number, a category name, and a specific monthly salary amount. All managed categories correspond to the various types of personnel within the company.

The company is divided into departments. Each department is characterized by a department number and a department name. It is managed by an employee who serves as the department head. An employee can be the head of only one department. Employees must belong to at least one department but may be part of several. There are no departments without employees.

The company manages projects, each identified by a project number, project name, start date, end date, and budget. Every project is led by an employee who serves as the project manager. A single employee may manage multiple projects. Each project must involve at least one employee.

Employees may or may not participate in one or more of these projects for varying durations. Projects can also be carried out in collaboration with partners, each identified by a partner number and name. All partners managed by the system are associated with at least one project. Each partner corresponds to a type (Part_type), identified by a number and a name. A partner can correspond to only one type. All listed types are in use.

Additionally, due to their legal status, partners may be related with one or more countries, although this is not mandatory. For example, a Franco-American partner is recorded as being affiliated with both France and the USA.

Countries are identified by a number and a name. Information about countries is gathered as needed.

Questions:

1. Provide the entity-relationship model.
2. Provide the corresponding relational model (Logical Data Model LDM).

Exercise 5

A company intends to create a database in order to computerize the management of its equipment. Each equipment is characterized by a serial number, a brand, a type, a weight, and the number of personnel required for its operation. The company requires that each evening, these pieces of equipment be stored.

A company intends to create a database to computerize the management of its equipment. Each equipment is characterized by a serial number, a brand, a type, a weight, and the number of personnel required for its operation. The company requires that each evening, this equipment be stored. To this end, it has several warehouses. Each warehouse is defined by a number, an address, a postal code, a locality, and a surface area. Every time a piece of equipment is stored in a warehouse, the date and time of arrival are recorded. Upon departure, the date and time of exit are also noted. All of the company's warehouses are capable of housing equipment; however, some equipment, due to their dimensions, cannot be stored and must remain on the construction sites.

Questions

1. Provide the entity-relationship model.
2. Provide the corresponding relational model.

Exercise 6

Provide the entity-relationship model and the corresponding relational model based on the following requirements:

1. The mechanic repairs a client's vehicle.
2. The vehicle may be of type car, truck, or other.
3. During the repair, the mechanic may detect anomalies of various types (mechanical, electrical, etc.) and may replace parts.
4. The mechanic records the date on which the repair was carried out.

Exercise 7

We want to design a database for managing a small library that offers a collection of books to its members. A book is described by a unique ISBN number, a title, an author, a publisher, and a publication date. The library holds one or more copies of each book. Each copy is identified by a

number and its purchase date. Each member can borrow multiple copies, and a member may borrow the same copy on multiple occasions. A loan is characterized by the loan date and the return date. Members are identified by a code and have a name and an address.

Questions

1. Provide the entity-relationship model.
2. Provide the corresponding relational model.

Exercise 8

Given the following relational schema:

Room (Snum, Sname, Capacity)
Session (Sid, StartTime, Price, Fid, Snum)
Film (Fid, Title, Year, Genre, Summary)

Questions:

1. Complete the above schema.
2. Answer with true or false and justify your answers:
   a) Can multiple films be projected in the same session?
   b) Can a film be projected in multiple sessions?

Exercise 9

Given the following logical relational schema:

Faculty (FNum, FName)
Department (DNum, DName, #EmployeeID, #FNum)
Teacher (EmployeeID, FirstName, LastName, Address, Phone, #DNum)
Student (StudentNum, FirstName, LastName, BirthDate, Address, #DNum)
Teaches (#StudentNum, #EmployeeID)

Questions:

Provide the corresponding Entity-Relationship Model for the above Relational Model.

Exercise 10

Consider the following relational schema for a library management:

Book (ISBN, Title, Publisher)
Loan (#MemberCode, #CopyNumber, LoanDate, ReturnDate)
Member (MemberCode, MemberName)
Copy (CopyNumber, #ISBN)

Questions

1. Answer True or False with justification:
   a. Can a single copy be borrowed by multiple members?
   b. Can a member borrow the same copy of a book on two different dates?
2. Provide the corresponding entity-relationship model.

Exercise 11

Given the database "grades_managment" described by the following relational schema:

Student (StudentID, SSN, FirstName, LastName, BirthDate, Gender, Cycle, #DeptNum)
Cycle ∈ {L1, L2, L3, M1, M2}
Teacher (TeacherID, SSN, FirstName, LastName, #DeptNum)
Course (CourseNum, CourseName, Cycle, #TeacherID, #DeptNum)
Grades (#StudentID, #CourseNum, Grade)
Department (DeptNum, DeptName)

SQL queries:

1. Write the Data Definition Language (DDL) instructions to create the Student relation, specifying integrity constraints (primary key, foreign key, domain constraint).
2. Provide the first name and last name of the students along with their grades in the "DB" course.
3. Find the names of teachers who have never taught the "DB" course.
4. Provide the number of students in each department.
5. Which teachers teach courses in departments different from the department to which they are assigned?

Supplementary exercises corrections

Exercise 1

1)

integrity constraint	type of constraint	field(s) associated
a. A constraint that ensures each car is associated with a driver	Referential	driverId
b. A constraint that guarantees that the driver in a race must be represented by a strictly positive integer indicating their rank	Domain Constraint	rank
c. A Constraint that guarantees a driver can have only one result per race.	Primary Key Constraint	(#raceId, #driverId)

2)

1. Go to the Database Tools tab.
2. Click on Relationships to open the relationship management window.
3. In the Relationships window, click on Show Table (or right-click and select this option).
4. Select the Driver and Car tables, then click Add.
5. Click Close to exit the table addition window.
6. Click on the DriverId field in the Driver table and drag it to the Driver field in the Car table.
7. An Edit Relationships dialog box will appear:
   a. Check that the connected fields are correct (they must both be driverId in the two tables).
   b. Check Enforce Referential Integrity (if you want to ensure that no car can be added without a corresponding driver).
   c. You can also check Cascade Update Related Fields and Cascade Delete Related Records if necessary.

8. Click Create to confirm the relationship.
   - A line will appear between the two tables, indicating the (1 - ∞) relationship between Driver and Car.
9. Save your changes by closing the Relationships window.

The relationship is now established, with driverId as a foreign key in the Car table pointing to driverId in the Driver table.

3) SQL queries

1. SELECT * FROM Competition;

2. SELECT DISTINCT Driver.driverId, driverName, phone
   FROM Driver, Resul
   WHERE Driver.driverId = Result.driverId
   AND rank = 1;

3. SELECT carId, Driver.driverId, driverName, brand
   FROM Car, Driver
   WHERE Voiture.idPilote = Pilote.idPilote;

4. SELECT raceId, AVG(duration) AS AverageTime
   FROM Result
   GROUP BY raceId;

5. SELECT Driver.driverId, AVG(Result.duration) AS AverageTime, COUNT(raceId) AS totalRaces
   FROM Driver, Result
   WHERE Driver.driverId= Resultat.driverIde
   GROUP BY Driver.driverId
   HAVING AVG(duration) < 100;

Exercise 2

1)

Statement	True	False
a. A cage can be located in multiple zones.		X
b. An animal belongs to multiple families.		X
c. An animal can receive the same vaccine on different dates.		X
d. An animal cannot receive multiple vaccinations		X

2)

Field:	CageNumber	CountOfidAnimal: AnimalID	
Table:	cage	Animal	
Total:	Group By	Count	
Sort:			
Show:	☑	☑	☐
Criteria:		>=2	
or:			

3)

SQL queries

1. SELECT animalName, countryOfOrigin
   FROM Animal;

2. SELECT DISTINCT countryOfOrigin
   FROM Animal;

3. SELECT animalName
   FROM Animal, Family
   WHERE Family.familyID = Animal.familyID
   AND familyName LIKE "A%";

4. SELECT cageNumber, COUNT(*) AS numberOfAnimals
   FROM Animal
   GROUP BY cageNumber;

5. SELECT cageNumber, COUNT(*) AS numberOfAnimalsG2
   FROM Animal
   GROUP BY cageNumber
   HAVING COUNT(*) >= 2;

CLIENT (ClientNumber, LastName, FirstName, Address)
SUPPLIER (SupplierCode, SupplierName)
CARD (CardNumber, Amount, SupplierCode)
LINE (LineNumber, ClientNumber, SupplierCode, Price, ActivationDate)

Exercise 3

Part B.2

a. SELECT LineNumber

   FROM LINE

   WHERE ActivationDate > "01-01-2025";

b. SELECT COUNT(*) AS Cards_Number

   FROM CARD

   WHERE Amount = 1000;

c. SELECT SUM(Amount) AS Total_Amount

   FROM CARD, SUPPLIER

   WHERE CARD.SupplierCode = SUPPLIER.SupplierCode

   AND SupplierName= "ALGCOM";

d. SELECT ClientNumber, COUNT(LineNumber) AS Lines_Number

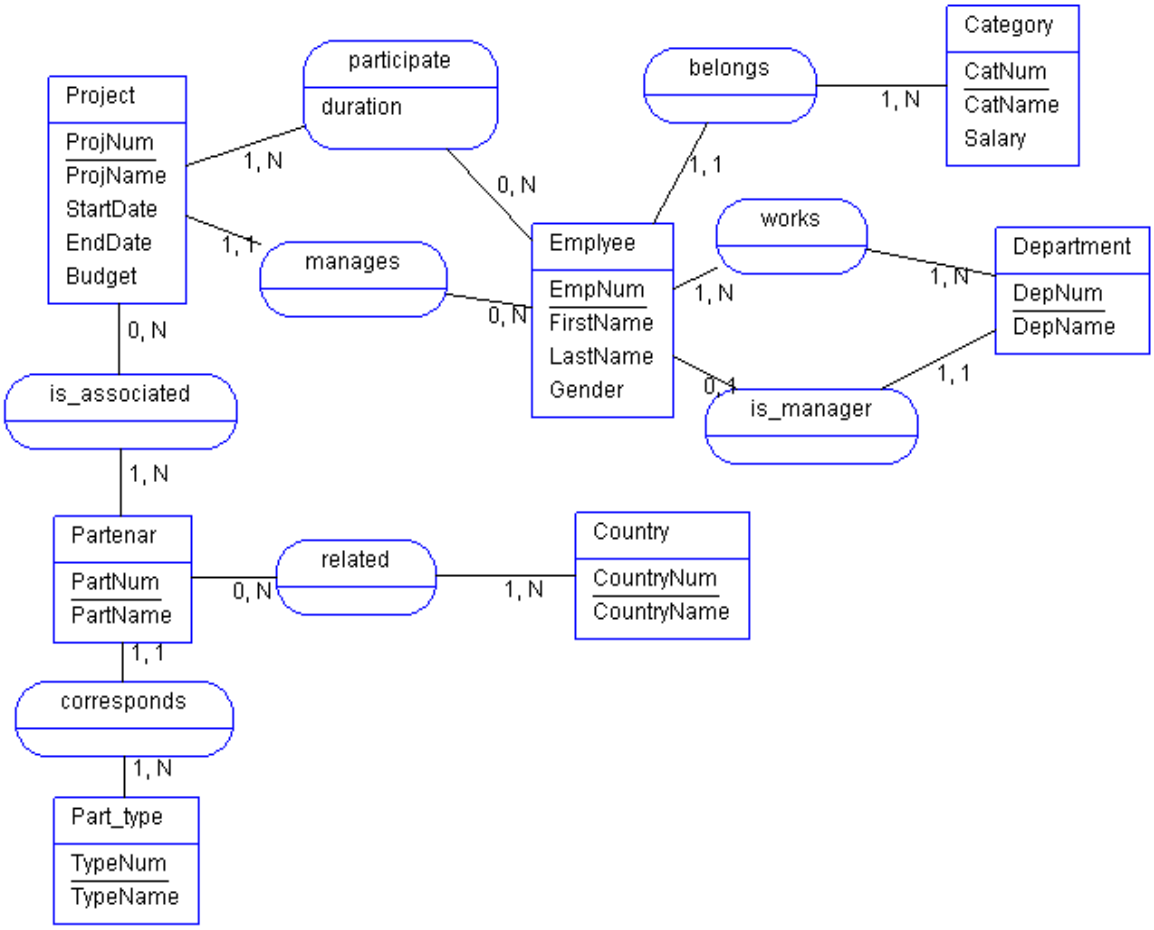
FROM LINE

GROUP BY ClientNumber

HAVING COUNT(LineNumber) >= 2;

Exercise 4

1. Entity-relationship model



2. Relational model

Employee (<u>EmpNum</u>, FirstName, LastName, Gender, #CatNum, #DepNum)

Category (<u>CatNum</u>, CatName, Salary)

Department (<u>DepNum</u>, DepName, #EmpNum)

Project (<u>ProjNum</u>, ProjName, StartDate, EndDate, Budget, #EmpNum)

Partenar (<u>PartNum</u>, PartName, #TypeNum)

Part_type (<u>TypeNum</u>, TypeName)

Country (<u>CountryNum</u>, CountryName)

related (#<u>PartNum</u>, #<u>CountryNum</u>)

is_associated (#<u>ProjNum</u>, #<u>PartNum</u>)

participate (#EmpNum, #ProjNum, duration)
works (#EmpNum, #DepNum)

Exercise 8

1.

Room (Snum, Sname, Capacity)
Session (Sid, StartTime, Price, #Fid, #Snum)
Film (Fid, Title, Year, Genre, Summary)
Primary keys are underlined.
Foreign keys are preceded by a #. (1 point)

2.

a) **False**, a single session cannot feature more than one film because each session has a unique identifier Sid, which cannot be duplicated.
b) **True**, a film can be shown in multiple sessions because the session relation includes a foreign key *Fid,* which can be repeated for each new session.

Exercise 11

SQL queries

1)

```
CREATE TABLE Student (
StudentID VARCHAR(10) PRIMARY KEY,
SSN VARCHAR(15) UNIQUE,
FirstName VARCHAR(30) NOT NULL,
LastName VARCHAR (30) NOT NULL,
BirthDate DATE,
Gender CHAR(1) CHECK IN ("F", "M"),
Cycle VARCHAR(2) CHECK IN ("L1", "L2", "L3", "M1", "M2"),
DeptNum VARCHAR(6) REFERENCES Department (DeptNum);
```

2)

```
SELECT FirstName, LastName, Grade
```

```
    FROM Student S, Course C, Grades G
    Where S.StudentID = G.StudentID
    AND C.CourseNum = G.CourseNum
    AND CourseName = "DB";
```

3)

```
    SELECT FirstName, LastName
    FROM Teacher
    WHERE TeacherID NOT IN (SELECT TeacherID
                            FROM Course
                            WHERE CourseName = "DB");
```

4)

```
  SELECT DeptNum, Count(StudentID)
  FROM Student
  GROUP BY DeptNum;
```

5)

```
  SELECT FirstName, LastName
   FROM Teacher T, Course C
   WHERE T.TeacherID= C.TeacherID
   AND T.DeptNum# C.DeptNum;
```

References for Further Reading

1. **Laudon, K. C., & Laudon, J. P. (2018).** Management Information Systems: Managing the Digital Firm (15th ed.). Pearson.
2. **O'Brien, J. A., & Marakas, G. M. (2017).** Introduction to Information Systems (16th ed.). McGraw-Hill Education.
3. **Rainer, R. K., & Cegielski, C. G. (2019).** Introduction to Information Systems: Enabling and Transforming Business (7th ed.). Wiley.
4. **Garcia-Molina, H., Ullman, J. D., & Widom, J.** (2008). *Database Systems: The Complete Book* (2nd ed.). Pearson.
5. **Codd, E.F.,** A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, 1970.
6. **Elmasri, R., & Navathe, S. B.** (2016). *Fundamentals of Database Systems* (7th ed.). Pearson.
7. **Date, C. J.** (2012). *Database Design and Relational Theory: Normal Forms and All That Jazz*. O'Reilly Media.
8. **Hernandez, M. J.** (2013). *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design* (3rd ed.). Addison-Wesley.
9. **Ramakrishnan, R., & Gehrke, J.** (2003). *Database Management Systems* (3rd ed.). McGraw-Hill.
10. **Silberschatz, A., Korth, H. F., & Sudarshan, S.** (2010). *Database System Concepts* (6th ed.). McGraw-Hill Education. ISBN: 978-0073523323.
11. **Stonebraker, M., & Hellerstein, J. M.** (2005). *Readings in Database Systems* (4th ed.). The MIT Press. ISBN: 978-0262693141.
12. **Abiteboul, S., Hull, R., & Vianu, V.** (1995). *Foundations of Databases*. Addison-Wesley. ISBN: 978-0201537710.