People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Oran Graduate School of Economics

Educational handout

Courses / Tutorials / Practical work

Databases and SQL: Practical Applications in Microsoft Access

For students of (2nd year / Preparatory cycle)

By Dr. Amel Mounia DJEBBAR

Academic year 2024 / 2025

Summary

CHAPTER 1 8
INTRODUCTION TO INFORMATION SYSTEMS 8
1.1. Concept of organization 9
1.2. Notion of system 9
1.3. Information system 10
1.4. Information 11
1.4.1. Definition 11
1.4.2. Roles of information 11
1.4.3. Sources of information 11
1.4.3.1. Internal Sources 11
1.4.3.2. External Sources 12
1.5. Information system functions 12
1.5.1. Information collect 12
1.5.2. Information storage 12
1.5.3. Information processing 12
1.5.4. Information distribution 12
1.6. Information system components 12
1.6.1. Information 13
1.6.2. Human resources 13
1.6.3. Material resources 13
1.6.4. Methods 13
CHAPTER 2 14
DATABASES AND DBMS 14
2.1. Database definition 15
2.2. Definition of a database management system 15
2.3. DBMS history 15
2.4. DBMS objectives 15
2.5. DBMS Architecture: 16
2.5.1. Single tier architecture 16
2.5.2. Two tier architecture 16
2.5.3. Three tier architecture 17

2.6. Levels of Abstraction in DBMS 17
2.6.1. Physical level 18
2.6.3. View level 18
CHAPTER 3 19
ENTITY–RELATIONSHIP (ER) MODEL 19
3.1. Generalities 20
3.2. Basic Concepts 20
3.2.1. Attribute 20
3.2.1.1. Definition 20
3.2.1.2. Properties 20
3.2.1.3. Attribute types 20
3.2.1.4. Attribute domain 21
3.2.2. Entity 21
3.2.2.1. Definitions 21
3.2.2.2. Entity identifier 22
3.3. Graphical Representation 22
3.4. Relationship and Cardinalities 22
3.4.1. Relationship 22
3.4.2. Cardinalities 22
3.4.3. Types of Cardinalities 23
3.4.4. Types of relationship 24
3.4.5. Relationship attributes 24
3.5. Reflexive relationship 25
3.5.1. Key Characteristics of a Reflexive Relationship 25
3.5.1.2. Same Entity Participation 25
3.5.1.3. Different Roles 25
3.5.1.4. Use Cases 25
3.6. N-ary relationships 26
3.6.1. Key Characteristics of N-ary Relationships 26
3.6.1.1. Multiple Entities Involved 26
3.6.1.2. N-ary Relationship 26
3.6.1.3. Context and Dependencies 27
3.6.1.4. Attributes in Relationships 27
3.7. Exercises 27

Exercise 1 27
Exercise 2 28
Exercise 3 28
Exercise 4 29
3.8. Solution of exercises 30
Solution of exercise 1 30
Solution of exercise 2 30
Solution of exercise 3 31
Solution of exercise 4 32
CHAPTER 4 33
RELATIONAL MODEL 33
4.1. Definition 34
4.2. Basic data structures 34
4.2.1. Attribute 34
4.2.2. Domain 35
4.2.3. Relation 35
4.2.4. Relationship Schema 36
4.2.5. Degree 36
4.2.6. Tuple 36
4.2.7. Primary key 36
4.2.8. Foreign key 37
4.2.9. Relational schema 38
4.3. Integrity Rules 38
4.3.1. Domain Integrity 38
4.3.2. Key Integrity 38
4.3.3. Referential Integrity 39
4.4. Rules for converting the entity-relationship model to the relational model (MLD) 40
4.4.1. Entity Types to Relations 40
4.4.2. Identifier as Primary Key 40
4.4.3. Relationships with Maximum Cardinality of 1 40
4.4.4. Relationships with Maximum Cardinality of N and 1 40
4.4.5. Relationships with Maximum Cardinality of N 41
4.5. Exercises 41
4.5.1. Exercise 1 41

4.5.2. Exercise 2 41
4.5.3. Exercise 3 41
4.5.4. Exercise 4 41
4.6. Solution of exercises 42
4.6.1. Solution of exercise 1 42
4.6.2. Solution of exercise 2 42
4.6.3. Solution of exercise 3 42
4.6.4. Solution of exercise 4 42
CHAPTER 5 43
SQL LANGUAGE 43
5.1. Definition 44
5.2. Data Definition Language (DDL) 44
5.2.1. Creating Database Objects 44
5.2.2. Altering Existing Database Objects 44
5.2.3. Deleting Database Objects 45
5.2.4. Creating Indexes 45
5.2.5. Defining Constraints 45
5.3. Data Manipulation Language (DML) 46
5.3.1. Inserting Data 46
5.3.2. Updating Data 46
5.3.3. Deleting Data 47
5.3.4. Retrieving Data 47
5.3.5. Aggregating Data 48
5.3.6. Sorting and Grouping Data 49
5.3.7. Using HAVING Clause with Grouping 50
5.3.8. Joins in SQL 51
5.3.8.1. Cartesian Product 51
5.3.8.2. Types of Joins 51
5.3.8.3. Self-Join 52
5.4. Exercise 52
5. 5. Solution of the exercise 53
CHAPTER 6 56
APPLICATION WITH ACCESS 56

6.1 Database Creation 57
6.2. Table Creation 57
6.2.1. Data Types 57
6.2.2. Field properties 58
6.2.3. Choice Lists 59
6.3. Entering Records in Datasheet View in Access 59
6.4. Create Request 60
6.4.1. Query design mode 60
6.4.2. SQL mode 61
6.5. Macro 63
6.5.1. Definition of a Macro 63
6.5.2. Key features of macros 63
6.5.3. Benefits of using macros 63
6.5.4. Steps to create macros in Access 63
6.6. Form in Microsoft Access 67
6.6.1. Definition of a Form 67
6.6.2. Key Features of Forms 67
6.6.3. Uses of Forms 67
6.6.4. Steps to create Form in Access 68
6.7. Report in Microsoft Access 71
6.7.1. Definition of a Report 71
6.7.2. Key Features of Reports 71
6.7.3. Uses of Reports 71
6.7.4. Steps to Create a Report in Access 72
6.8. Case study 74
6.9. Solution if the case study 76
Conclusion 99
References 100

List of Figures

Figure 1.1. A car assembly factory 9
Figure 1.2. Organizational information system 10
Figure 2.1. Single tier architecture 16
Figure 2.2. Two tier architecture 16
Figure 3.2. Two tier architecture 17
Figure 6.1. Creation of new DB 57
Figure 6.2. The different types of data. 58
Figure 6.3. Field properties. 58
Figure 6.4. Choice lists. 59
Figure 6.4. Entering records in Datasheet view mode. 60
Figure 6.5. Creating a query in Design mode. 61
Figure 6.6. Creating a query in SQL mode. 62
Figure 6.7. Creating Macro. 64
Figure 6.8. Macro actions in Access 65
Figure 6.9. Create Form in Access 69
Figure 6. 10. Create Form in Access 70
Figure 6.11. Create Report in Access 72
Figure 6.12. Report design view 73

CHAPTER 1

INTRODUCTION TO INFORMATION SYSTEMS

1.1. Concept of organization

The concept of organization involves structuring work groups, defining their composition, and coordinating their activities. It focuses on establishing organizational links that facilitate collaboration between individuals and teams, ensuring alignment with common goals. Effective organization enhances efficiency, communication, and overall productivity by creating a well-structured framework for achieving objectives.

1.2. Notion of system

A system is a structured set of material or immaterial elements, such as people, machines, methods, and rules interconnected through relationships. These elements work together within a defined process to transform inputs into outputs, see Figure 1.1. The efficiency and functionality of a system depend on the interactions between its components, the processes governing them, and the intended objectives.

Example:

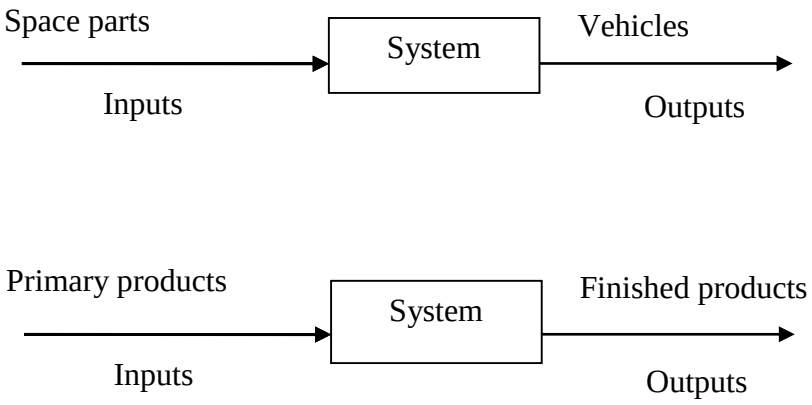


Figure 1.1. A car assembly factory

The company needs a decision-making system to achieve its objectives. This system is called the control system or decision system, and its role is to:

- Decide and manage the organization of the system.
- Process the information circulating in the system.
- Implement action plans for the operating system.
- Ensure that actions are consistent with management's strategic objectives.

The system corresponding to the company's activity (flow transformation) is called the operating system (OS), and its role is to:

- Receive decisions sent by the control system.
- Carry out action plans decided by the control system.
- Send information back to the control system for control.

As the amount and complexity of information exchanged between these two systems increases, another system is needed to store and process this information more efficiently. This system is called the Information System (IS) [1].

Figure 2 describes the three systems and the relationships between them in terms of communication.

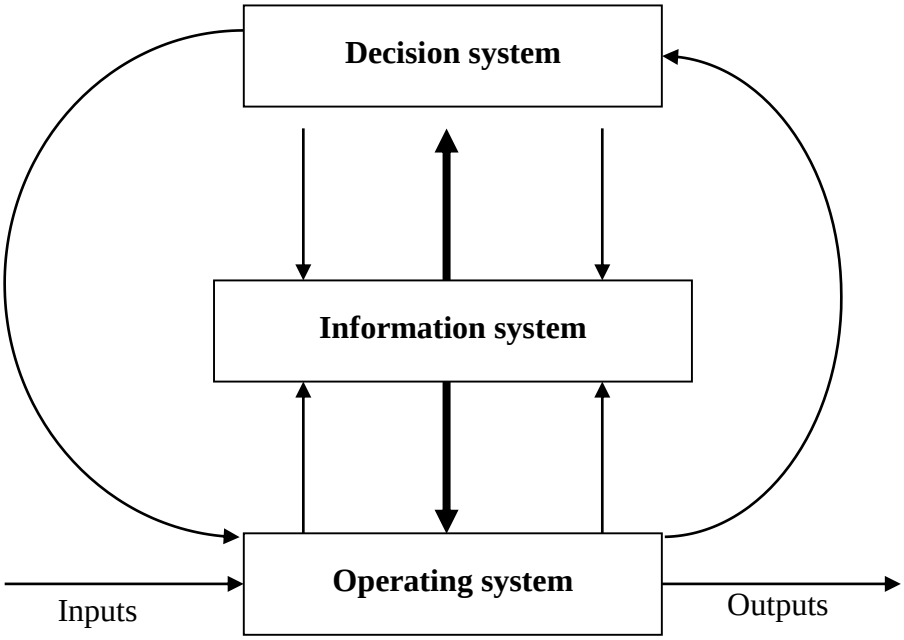


Figure 1.2. Organizational information system

1.3. Information system

An information system consists of interconnected elements, including employees, computers, rules, and methods designed to collect, store, process, and distribute information. It plays a crucial role in supporting the operational system by ensuring that relevant data is efficiently managed and made available to the control system. By facilitating decision-

making, coordination, and communication, an information system enhances the overall effectiveness and adaptability of an organization.

1.4. Information

1.4.1. Definition

(1) A number of data grouped together and relating to the same context give rise to information.

(2) Information is any coded written, sound, visual or audiovisual information that can be stored or transmitted in order to modifying the behavior of a process.

Data is an elementary description of information.

Information = data + meaning (depends on each individual)

1.4.2. Roles of information

- Information is what modifies the perception of a situation.
- Information reduces uncertainty.
- Information is used to make a decision.
- Information is relative to the recipient: what is information for one is not necessarily information for another.

1.4.3. Sources of information

Information originates from both internal and external sources, each playing a crucial role in decision-making and organizational efficiency.

By integrating both internal and external information, organizations can make informed decisions, optimize strategies, and enhance overall performance.

1.4.3.1. Internal Sources

These include data generated within the organization, such as reports, financial records, employee feedback, and operational metrics. Internal sources provide valuable insights into performance, efficiency, and internal processes.

1.4.3.2. External Sources

These encompass data from outside the organization, such as market research, customer feedback, industry trends, government regulations, and competitor analysis. External sources help organizations stay competitive and adapt to changing environments.

1.5. Information system functions

The functions of the information system [2] are to collect, store, process and distribute information.

1.5.1. Information collect

To function, the system must be supplied with information from a variety of internal and external sources.

1.5.2. Information storage

Once the information has been entered, we need to ensure its continuity, i.e. guarantee that it is stored in a durable and viable way, using the resources provided by computer disks (storage media). However, paper is still widely used in business (paper archives).

1.5.3. Information processing

To be usable, information has to be processed. Processing can be manual (rarely used) or automatic (carried out by computers). The main types of processing consist in searching for and extracting information, consolidating and comparing information, modifying and deleting information, or generating new information by applying calculations.

1.5.4. Information distribution

In order to be used, information must reach its recipient as quickly as possible. There are many ways of distributing information: on paper, orally and, more and more often, using digital media, which guarantee optimum transmission speed and the possibility of reaching as many people as possible.

1.6. Information system components

The components of an information system [1] are as follows:

1.6.1. Information

All information, whatever its form, is part of the IS. However, in the field of management, only formalized information (of natural or technical origin) is truly operational.

1.6.2. Human resources

Human resources are made up of all the people who receive, manipulate and transmit information. Example: All the people in a company: users, decision-makers, etc.

1.6.3. Material resources

Material resources consist of all the machines used to receive, manipulate and transmit information. Example: photocopiers, scanners, computers, telecommunications, etc.

1.6.4. Methods

Methods are all the working tools and rules used to solve management problems. Example: models (mathematical, operational research, accounting, economic, etc.), computer programs, computer software, etc.

CHAPTER 2

DATABASES AND DBMS

2.1. Database definition

A database is a structured collection of data that is organized and stored for efficient retrieval, management, and manipulation using specialized software. It serves as a centralized repository for storing, organizing, and managing information in a systematic way.

2.2. Definition of a database management system

A Database Management System (DBMS) is a software program that allows you to: describe, modify, interrogate and administer the data in a database.

2.3. DBMS history

A Database Management System evolved from early hierarchical and network models in the 1960s to modern relational and NoSQL systems [3], revolutionizing data storage and retrieval in various industries.

- Database Management Systems (DBMS) emerged in the 1960s as a solution to handle large volumes of data efficiently.
- The CODASYL model, developed in the late 1960s, was an early attempt at standardizing database systems.
- IBM's Information Management System (IMS) and the Integrated Data Store (IDS) were among the first commercially successful DBMS in the 1960s and 1970s.
- Edgar Codd's relational model, introduced in 1970, laid the foundation for modern relational database management systems (RDBMS).
- Oracle, founded in 1977, became a prominent player in the 1980s, popularizing the use of SQL and relational databases.
- In the 21st century, NoSQL databases and distributed systems have gained traction, providing alternatives to traditional RDBMS for handling diverse data types and scalability challenges.

2.4. DBMS objectives

The objectives of a DBMS can be summarized in the following points:

- Organize and structure data for efficient storage and retrieval.
- Ensure data integrity, accuracy, and consistency through constraints.

- Implement security measures to control access and protect data confidentiality.
- Manage concurrent access to prevent conflicts and maintain data consistency.
- Provide data independence, allowing changes to the database structure without affecting applications.
- Optimize performance, support queries, and enable scalability for growing data and user demands.

2.5. DBMS Architecture:

DBMS architecture can be classified into three types according to the use and requirements of the users [4]:

2.5.1. Single tier architecture

Single-tier architecture is an architecture in which the entire application resides on the user's machine.

Figure 2.1. Single tier architecture

In this type of architecture, the database is readily available on the client machine; any request made by the client does not require a network connection to be processed on the database.

2.5.2. Two tier architecture

Two-tier architecture is a software architecture in which a presentation layer or interface runs on the client, and a data layer or data structure is stored on the server.

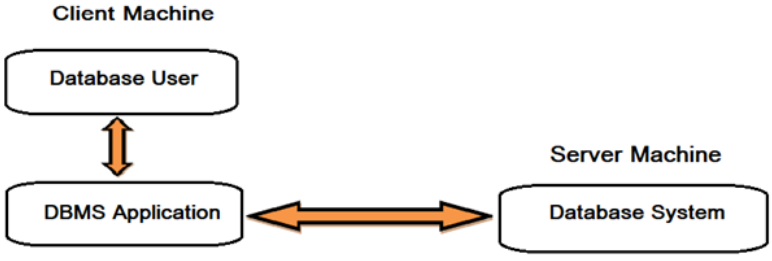


Figure 2.2. Two tier architecture

In two-tier architecture, the database system exists on the server machine and the DBMS application exists on the client machine, both these machines are connected to each other through a trusted network. Whenever the client machine requests to access the database present on the server using a query language like SQL, the server performs the request on the database and returns the result to the client.

2.5.3. Three tier architecture

A multi-tier architecture consists of three layers: client layer, business layer, and data layer. Its major advantage is improved scalability, as the application server can be deployed on multiple machines.

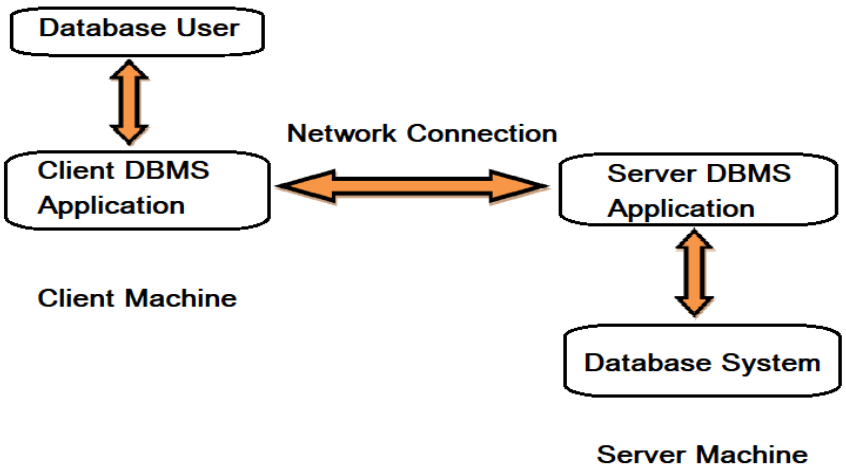


Figure 3.2. Two tier architecture

In multi-tier architecture, there is another layer between the client machine and the server machine. In this architecture, the client application does not communicate directly with the database system present on the server machine, rather the client application communicates with the server application and the server application internally communicates with the database system present on the server.

2.6. Levels of Abstraction in DBMS

Data abstraction is the property by which we hide non-essential functionality from clients. Application developers use abstraction to make the system efficient in terms of data retrieval and reduce complexity in terms of user-friendliness [5]. Data abstraction is used for the following purposes:

- To provide an abstract view of data.

- To hide complexity from users.
- To simplify user interaction with the DBMS.

The different levels of data abstraction in the DBMS are as follows:

2.6.1. Physical level

The lowest level of abstraction describes how data is actually stored.

2.6.2. Logical level

The next level of abstraction describes what data is stored in the database and what relationship exists between this data.

2.6.3. View level

The highest level of data abstraction describes only part of the overall database. It simplifies interaction with the user and it provides many views or multiple views of the same database.

CHAPTER 3

ENTITY–RELATIONSHIP (ER) MODEL

3.1. Generalities

This is a conceptual model designed in the 1970s that resulted from the work of BACHMAN, CHEN, TARDIEU [6]. It is used for the initial design phase. It uses a graphical representation for data modeling.

3.2. Basic Concepts

3.2.1. Attribute

3.2.1.1. Definition

An attribute is defined as the field or the smallest unit of data with a name.

Example: Name, first name, date of birth, car registration number, company name, ...

3.2.1.2. Properties

Simple attribute

An attribute that cannot be decomposed into other attributes. Example: Name, First Name, ID Number, ...

Composite attribute

The value of the attribute is a concatenation of the values of several simple attributes. Example: Address (Street, Postal Code, City).

Derived attribute

The value of the attribute is calculated or derived from the values of other attributes. Example: Age, Average, ...

Null value (NULL)

For an attribute, it is an undefined value. Example: For a customer whose date of birth is unknown, the date of birth attribute takes the value NULL.

3.2.1.3. Attribute types

The attribute type is crucial in the design of databases, information systems, or data models, as they define the nature of the data that can be stored and manipulated.

Integer

This attribute type is used to represent whole numbers, i.e., numbers without a decimal part. Example: Number of items in stock, Unique user IDs, Age of a person, …

Real

This type represents real numbers, including numbers with decimal points. It is used for values that require precision, such as prices or measurements. Example: Product prices, Temperature readings, Distances, ….

Date

A date attribute is used to store a specific date and sometimes time. This type is essential for managing anything related to time. Example: Birthdate, Account creation date, Transaction date and time.

String

This type represents a sequence of characters. It is used to store text, such as names, addresses, or descriptions. Example: Username, Email address, Product description

3.2.1.4. Attribute domain

Refers to the set of permissible or admissible values that an attribute (or a group of attributes) can have databases. It defines the valid data that can be stored in a particular attribute (e.g., column in a database table), ensuring consistency, accuracy, and proper interpretation of data.

Examples:

- Age attribute domain: integer values from 0 to 120.
- Gender attribute domain: predefined string values like 'male', 'female',
- If the price of products is between 200 DA and 500 DA, then the domain of the price attribute is [200… 500].

3.2.2. Entity

3.2.2.1. Definitions

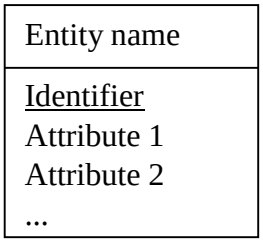
- An entity is an object in the discourse universe = (subject, theme).
- An entity type conceptually defines an entity whose members all share the same characteristics. For example, "Customer" or "Product" in a business database.
- An entity occurrence is composed of the set of values of each attribute of an entity type. Example or record within that entity type, such as a particular customer or a specific product.

3.2.2.2. Entity identifier

An entity identifier, also known as a primary key, is a specific attribute (or a combination of attributes) in an entity type that ensures each record is unique. In other words, it helps distinguish one instance of an entity from another, preventing ambiguity when accessing or referencing specific data in a database or system.

3.3. Graphical Representation

A graphical representation is a visual tool used in database design to represent the structure and relationships between entities within a system. An ER diagram is fundamental in conceptual data modeling and helps communicate how different entities (objects, people, or things) relate to each other. Here's an in-depth look at the components and how they are graphically represented in an ER diagram.

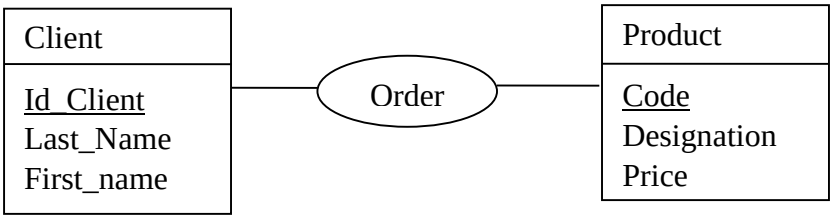


3.4. Relationship and Cardinalities

Relationship

A relationship in data modeling or database design refers to a logical connection or association between two or more entities. Each entity involved in the relationship plays a specific, well-defined role, contributing to the overall interaction or dependency between them.

Example: Clients order products.



Cardinalities

Refers to the rules governing how entities are associated in a relationship, specifying how many instances of one entity can be related to instances of another entity. It is defined

by a pair of numbers (min, max) that represent the minimum and maximum number of times an entity can participate in the relationship.

Minimum Cardinality (min):

This defines the minimum number of instances of one entity that can be related to another. It indicates whether participation is optional (min = 0) or mandatory (min > 0).

min: corresponds to the answer to the following question:

- How many times is at least one entity from A related to an entity from B?

Maximum Cardinality (max)

This specifies the maximum number of instances of one entity that can participate in the relationship with the other entity. It determines whether an entity can be related to only one instance (max = 1) or multiple instances (max > 1).

max: corresponds to the answer to the following question:

- How many times at most is an entity from A related to an entity from B?

These questions should be asked in both directions from A to B and from B to A.

Types of Cardinalities

Based on the values of (min, max), cardinality can describe different relationship types, such as:

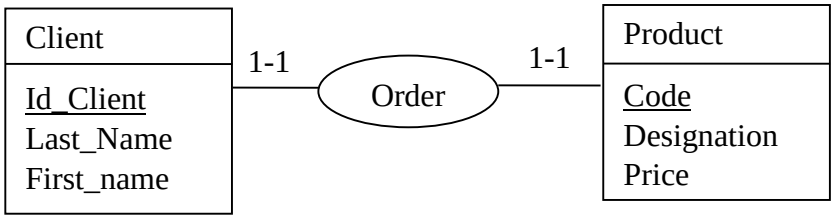
One-to-One (1, 1): One instance of an entity is related to only one instance of another entity (e.g., a person has one passport, and one passport belongs to one person).

One-to-Many (1, N): One instance of an entity is related to multiple instances of another entity (e.g., a department can have multiple employees, but each employee belongs to one department).

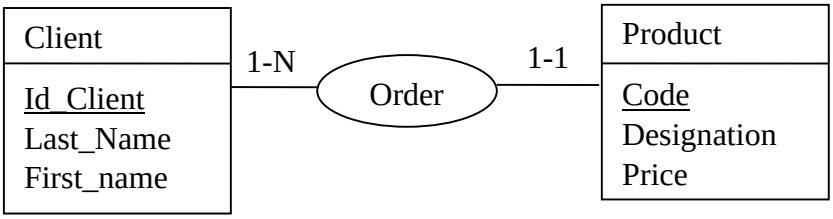
Many-to-Many (M, N): Multiple instances of an entity are related to multiple instances of another entity (e.g., students can enroll in many courses, and each course can have many students).

Types of relationship

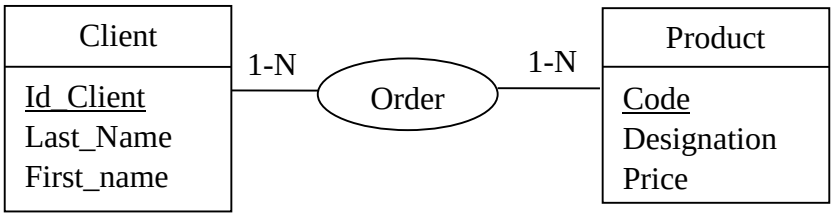
Relationship 1-1: A given client only orders one product. A product is ordered by only one client.



0-N or 1-N Relationship: A given client orders multiple products. A product is only ordered by a single client.



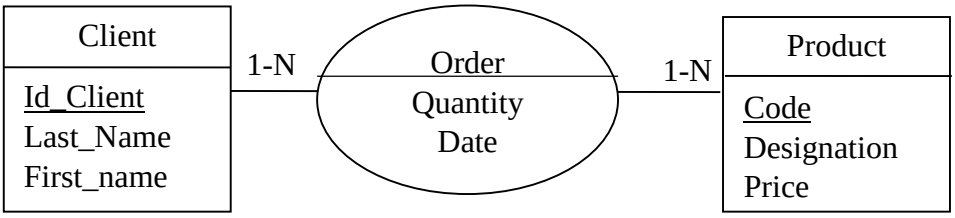
Relationship M-N: A given client orders multiple products. A given product is ordered by multiple clients.



Relationship attributes

In a M-N relationship, it is possible to characterize the relationship using attributes.

Example: An order is placed on a given date and concerns a fixed quantity of product.



3.5. Reflexive relationship

A reflexive relationship (also known as a recursive relationship) is a type of relationship in which an entity is related to itself. In this scenario, instances of the same entity type participate in the relationship by playing different roles, establishing a connection between them within the entity itself.

3.5.1. Key Characteristics of a Reflexive Relationship

3.5.1.2. Same Entity Participation

Both sides of the relationship involve instances of the same entity. This means that an entity can be related to other instances of the same entity, rather than being connected to instances of a different entity.

3.5.1.3. Different Roles

In a reflexive relationship, the participating instances of the entity often play distinct roles. For example, one instance might be a "manager," while the other is an "employee" within an organization, both of which are part of the same "Person" entity.

3.5.1.4. Use Cases

Reflexive relationships are common when modeling hierarchies, organizational structures, or part-whole relationships, where elements of the same type relate to each other in a specific manner.

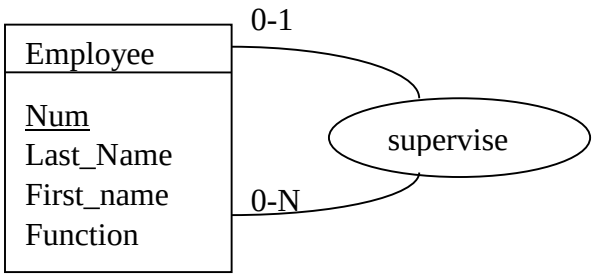
Example: Employee-Manager Relationship

In an organization, employees can have supervisors or managers, who are also employees. The relationship between "Employee" and "Manager" is reflexive because both roles involve instances of the same "Employee" entity.

- Entity: Employee
- Relationship: "Supervise" (reflexive)

- Role: One employee supervises (manager), and another employee is supervised (subordinate).

This relationship shows that within the same entity "Employee," there is a link between employees based on their organizational roles.



3.6. N-ary relationships

A relationship can involve more than two entities, and such relationships are known as n-ary relationships or ternary (three entities) and higher-order relationships. These relationships capture associations where multiple entities are involved simultaneously, reflecting complex real-world interactions that cannot be fully described by binary (two-entity) relationships.

3.6.1. Key Characteristics of N-ary Relationships

3.6.1.1. Multiple Entities Involved

Unlike binary relationships, which only involve two entities, an n-ary relationship connects three or more entities simultaneously. This allows for the representation of complex associations where the involvement of multiple entities is necessary for the relationship to exist.

3.6.1.2. N-ary Relationship

The "n" in n-ary signifies the number of entities involved in the relationship. For instance:

- A ternary relationship involves three entities.
- A quaternary relationship involves four entities, and so on.

3.6.1.3. Context and Dependencies

These relationships express dependencies between several entities, where the meaning of the relationship depends on all participating entities. For example, a supplier providing a specific product to a specific store at a particular time is a scenario that might require a ternary relationship involving "Supplier," "Product," and "Store."

3.6.1.4. Attributes in Relationships

Often, n-ary relationships will have their own attributes that provide additional information about the relationship. For example, in a relationship between "Student," "Course," and "Semester," an attribute like "Grade" may describe the student's performance in the course for that particular semester.

Example:

Consider a scenario where three entities—Supplier, Product, and Store—are involved. The relationship describes which suppliers provide which products to which stores.

- Entities: Supplier, Product, Store
- Relationship: "Supplies"

In this ternary relationship:

- A supplier can supply multiple products.
- A product can be supplied to multiple stores.
- A store can receive products from multiple suppliers.

The relationship "Supplies" depends on all three entities for its full meaning. This means the relationship only exists if all three entities participate simultaneously—i.e., a supplier provides a particular product to a particular store.

3.7. Exercises

Exercise 1

Consider representing the activities of a library that has a collection of books available to its subscribers. Each book in the library is described with a unique code, a title, an author, a publisher, and a publication date.

The registration of a new subscriber consists of entering their number, which will serve as an identifier, their last name and first name, their address, and their phone number.

Each subscriber can borrow multiple books. For each loan, the book code, the subscriber's number, and the loan date are recorded. Upon the return of the book, the return date is recorded.

Question: Provide an Entity-Relationship model for this scenario. Note the cardinalities between entities and underline the identifiers of the entities.

Exercise 2

One of Oran's hotels wanted to set up a database to better manage its clientele. To this end, the hotel manager has collected the following set of information:

- Customers are identified by number, surname, first name and phone number.

- Each room has a number, a type and a surface area. We also need to know whether the room has a sea view or not. The room type is identified by a number and characterized by the number of people who can occupy the room.

- When the customer makes a reservation, the name of the chosen formula is saved, along with the room number. Each reservation is characterized by a number, the type of board (breakfast, half-board or full-board) and the period (start and end dates).

Question: propose an Entity/Association model that meets the requirements described above.

Exercise 3

The personnel management service of a company wishes to acquire a tool that allows them to digitally manage their employees, their salaries, and their leave. The following specifications have been produced from a need analysis conducted with the personnel service.

Every employee is identified by his social security number, a last name, a first name, and a date of birth.

Every employee has a function and belongs to a department. An identifier and a name characterize functions and departments.

For each employee, the hire date is managed.

Salary Management: For each employee, a salary is assigned.

Leave Management: For each employee, every taken leave is recorded (assuming that leave always involves one or more full days, for which the date and duration are retained). A number and a type characterize a leave.

Question: Give an Entity/Relationship model.

Exercise 4

We aim to computerize the organization of a series of conferences. The various conferences take place in different universities on different dates and are organized by different people.

Each conference consists of a set of presentations. It is identified by a name and the name of the university where it is held and is described by the date on which it takes place. A university is identified by its name and characterized by its address.

Each presentation is identified by a unique title and is characterized by a summary.

A presentation is given by only one speaker in a conference. Multiple speakers can present the same presentation in different conferences.

We want to keep track in the database of participants' registrations for the various conferences. Organizers and speakers of a conference are considered participants in that conference. A speaker can also be an organizer. A person can participate in organizing only one conference.

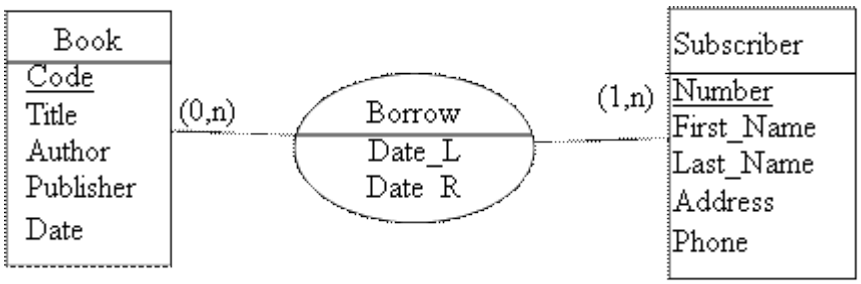
A participant is identified by their ID and described by their name, first name, and address.

For each speaker, we also want to record the name of the institution they come from, and for each organizer, we also record a phone number.

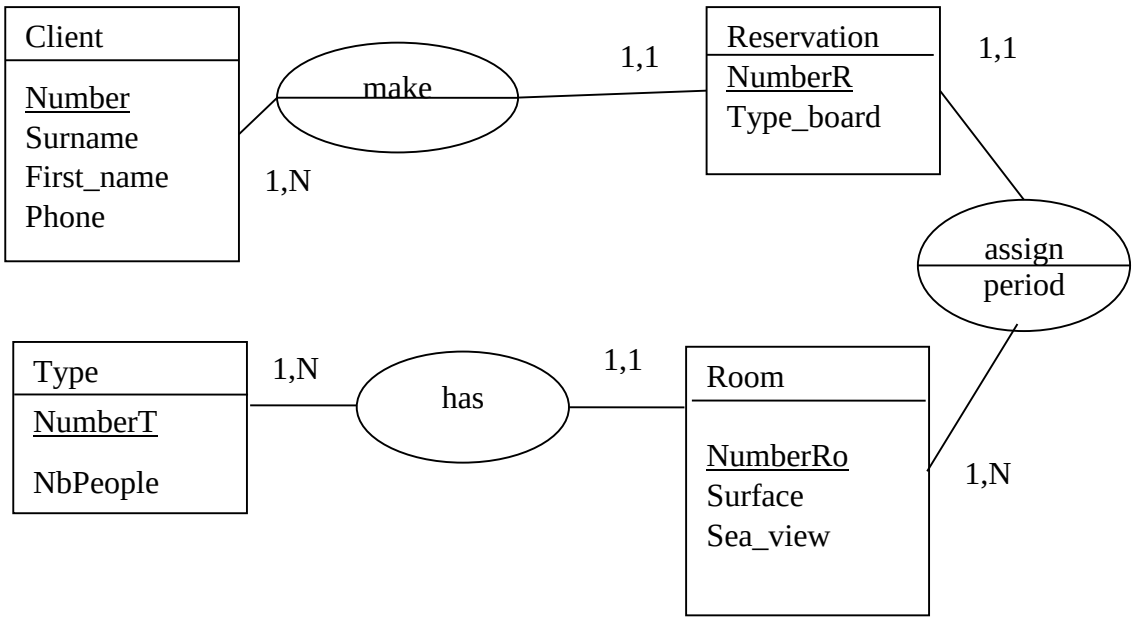
Question: Provide the entity-relationship diagram for the database described above.

3.8. Solution of exercises

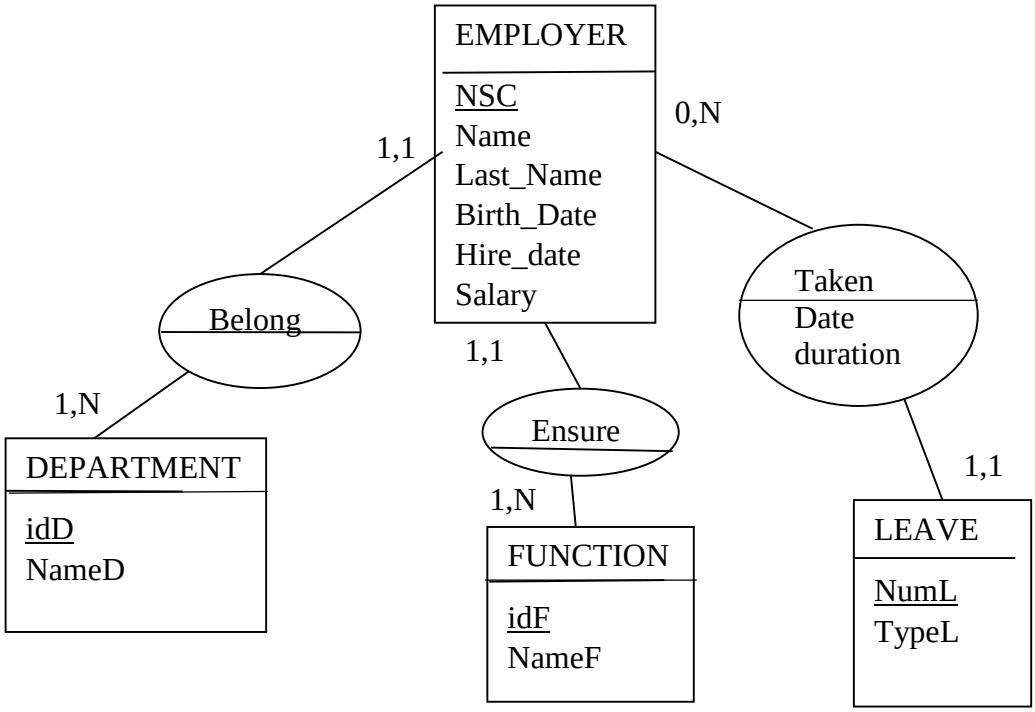
Solution of exercise 1



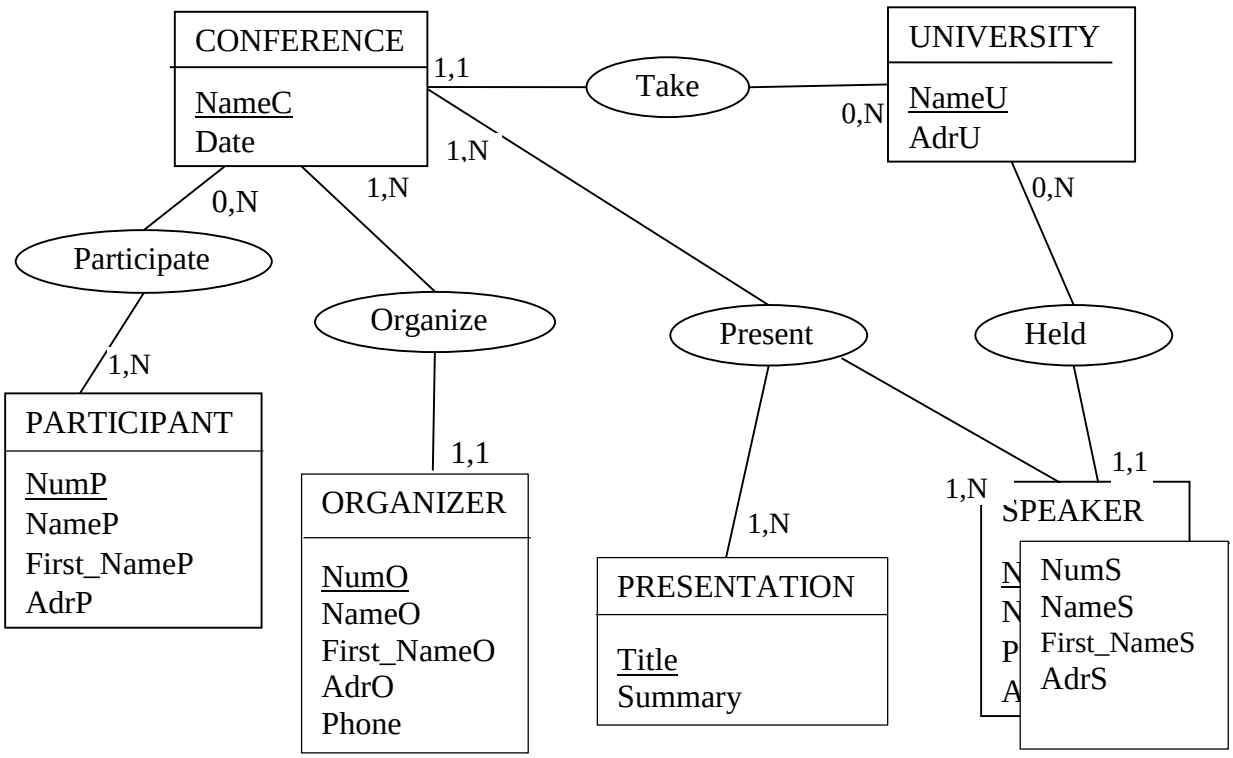
Solution of exercise 2



Solution of exercise 3



Solution of exercise 4



CHAPTER 4

RELATIONAL MODEL

4.1. Definition

The Relational Model is the basis for most modern Database Management Systems (DBMS). In this model, data is organized into tables, known as relations, where each table consists of rows and columns. Each row (or tuple) represents a unique data record, and each column (or attribute) represents a data field. The relational model enables data to be stored in a structured, highly organized manner and supports efficient querying and manipulation through relational algebra.

This model relies on two core components:

- **Data Definition Language (DDL)**: DDL is used to define and modify the database structure or schema. It includes commands to create, alter, and delete database objects like tables, indexes, views, and constraints. Examples of DDL commands include CREATE, ALTER, and DROP.
- **Data Manipulation Language (DML)**: DML is used for accessing and modifying data stored within the database tables. It includes commands that allow users to retrieve, insert, update, and delete data. Common DML commands include SELECT, INSERT, UPDATE, and DELETE.

The relational model provides a foundation for high data integrity, consistency, and support for complex querying using SQL, making it widely adopted across various industries and applications. This model also facilitates relationships between tables using foreign keys, which enables a structured way of linking data and maintaining referential integrity.

4.2. Basic data structures

4.2.1. Attribute

In a Relational Database, an attribute is a named property or characteristic of an entity that represents a specific type of data stored in the database. In simple terms, an attribute corresponds to a column in a table, where each column represents a particular aspect or detail about the entities described by the table.

Each attribute has a name, this is the label given to the attribute, describing the type of data it represents, such as Name, Age, or SocialSecurityNumber.

Attributes are fundamental in structuring data in a relational model because they define the characteristics of each data record (or row) in a table. Each attribute stores values

specific to one aspect of an entity, allowing for a structured and organized representation of information.

4.2.2. Domain

In the Relational Model of databases, the domain of an attribute refers to the set of all possible values that an attribute (or column) can hold. Each attribute in a relational database has a specific domain that defines the type of data it can store, ensuring data consistency and integrity within the database. Domains can be finite or infinite, depending on the attribute's requirements.

For example:

- The attribute Name may have a domain consisting of all possible combinations of letters and characters (often referred to as a string data type).
- The attribute Age might have a domain defined as all positive integers within a specified range (e.g., 0–120).
- The attribute BirthDate may have a domain constrained to valid date values.

4.2.3. Relation

A relation in the relational model is defined as a subset of the Cartesian product of multiple domains. Formally, if we have domains D_1, D_2,…, D_n, then a relation R is a subset of the Cartesian product $D_1 \times D_2 \times \ldots \times D_n$. Each domain D_i represents the set of possible values for an attribute of R, and n represents the degree of the relation (i.e., the number of attributes).

Notation:

$R \subseteq D_1 \times D_2 \times \ldots \times D_n$, where D_1, D_2, …,D_n are the domains associated with relation R.

Example:

Consider the domains:

STUDENT_NAME = {Absar, Madani}

STUDENT_FIRST_NAME = {Amina, Mohamed, Mustapha}

DATE_OF_BIRTH = {dates between 1/1/1990 and 31/12/2010}

An example instance of the STUDENT relation could be:

STUDENT = {(Absar, Amina, 1/1/1992), (Madani, Mohamed, 2/2/1994)}

4.2.4. Relationship Schema

A relationship schema defines the structure of a relation by specifying its name and attributes. Each attribute has a name and an associated domain, which restricts the type of data it can hold, ensuring data consistency. Example: STUDENT (STUDENT_NAME, STUDENT_FIRST_NAME, DATE_OF_BIRTH).

4.2.5. Degree

In the relational model, the degree of a relation (or relationship) is defined as the number of attributes it contains. Each attribute represents a specific characteristic or property of the entities stored within the relation.

- A degree of 1 signifies a unary relation, with a single attribute.
- A degree of 2 signifies a binary relation, involving two attributes.

Similarly, relations can have higher degrees, with each additional attribute increasing the degree.

Example:

For a relation schema STUDENT (STUDENT_NAME, STUDENT_FIRST_NAME, DATE_OF_BIRTH):

The degree is 3, as it contains three attributes (STUDENT_NAME, STUDENT_FIRST_NAME, DATE_OF_BIRTH).

4.2.6. Tuple

A tuple is a single row within a relational table (or relation) that represents a unique record or instance of the data defined by that table. Each tuple consists of a set of values, one for each attribute in the relation, and collectively these values describe a specific entity or object within the context of the relation.

4.2.7. Primary key

A primary key is a specific attribute (or a combination of attributes) in a relational table that uniquely identifies each tuple (or row) within that relation. The primary key ensures that no two tuples can have the same value for the key attribute(s), thereby maintaining the integrity and uniqueness of records in the database.

Example:

Consider the relation STUDENT with the schema:

STUDENT (STUDENT_ID, STUDENT_NAME, STUDENT_FIRST_NAME, DATE_OF_BIRTH)

Here, STUDENT_ID could serve as the primary key:

STUDENT = {(1, Absar, Amina, 1/1/1992), (2, Madani, Mohamed, 2/2/1994)}

In this example, STUDENT_ID uniquely identifies each student in the table, ensuring that the records for Absar and Madani are distinct and easily accessible. The primary key plays a crucial role in maintaining the structure and reliability of the database, facilitating efficient data retrieval and relationships between different tables.

4.2.8. Foreign key

A foreign key is an attribute (or a set of attributes) in a relational table that establishes a link between that table and a primary key in another table. It serves as a reference that maintains the relational integrity between the two tables, allowing for the creation of meaningful relationships within the database.

Example:

Consider two tables: STUDENT and COURSE_ENROLLMENT.

STUDENT table:

STUDENT (STUDENT_ID, STUDENT_NAME, STUDENT_FIRST_NAME, DATE_OF_BIRTH)

Where STUDENT_ID is the primary key.

COURSE_ENROLLMENT table:

COURSE_ENROLLMENT (ENROLLMENT_ID, STUDENT_ID, COURSE_CODE)

In this case, STUDENT_ID in the COURSE_ENROLLMENT table serves as a foreign key referencing the STUDENT_ID primary key in the STUDENT table.

Data Example:

STUDENT = { (1, Absar, Amina, 1/1/1992), (2, Madani, Mohamed, 2/2/1994) }

COURSE_ENROLLMENT = { (101, 1, 'CS101'), (102, 2, 'CS102') }

Here, the foreign key STUDENT_ID in COURSE_ENROLLMENT references the primary key STUDENT_ID in STUDENT, indicating which student is enrolled in which course.

4.2.9. Relational schema

A relational schema is a blueprint that outlines the structure of a relational database. It is composed of a set of relation schemas, each of which defines a table within the database. Each relation schema specifies the name of the relation (table) and the attributes (columns) that it contains, along with their respective data types and constraints.

4.3. Integrity Rules

Integrity rules in relational databases are essential guidelines that ensure the accuracy and consistency of the data within the database. These rules enforce structural constraints on the tuples (rows) that compose the relations (tables), maintaining the reliability of the database. The main integrity rules include:

4.3.1. Domain Integrity

Domain integrity ensures that all values in a specific column (attribute) of a relation adhere to the defined domain for that attribute. This means that the values must conform to the specified data type and any additional constraints (such as length or format). For example, if an attribute is defined as an integer, any entry in that column must be an integer. If the domain for DATE_OF_BIRTH is defined to allow dates only between 1/1/1990 and 31/12/2010, any date outside this range would violate domain integrity.

4.3.2. Key Integrity

Key integrity mandates that primary key values must be unique across the relation and cannot contain NULL values. This uniqueness guarantees that each tuple (record) in the relation can be distinctly identified.

Tuple Uniqueness: For instance, if the STUDENT_ID is designated as the primary key in a STUDENT table, every STUDENT_ID must be unique and not NULL. This ensures that there cannot be two students with the same identifier, thereby enforcing the uniqueness of each student record.

4.3.3. Referential Integrity

Referential integrity dictates that the values of foreign keys in one relation must match existing values of the primary key in the referenced relation. This rule enforces valid links between tables, preventing orphaned records in the database.

Dependent Relationships: For example, in the COURSE_ENROLLMENT table, if STUDENT_ID is a foreign key that references STUDENT_ID in the STUDENT table, every STUDENT_ID value in COURSE_ENROLLMENT must correspond to an existing STUDENT_ID in the STUDENT table. This ensures that a course enrollment record cannot exist for a student who is not represented in the STUDENT table.

Example:

Client

ClientNumber	Name	First name	City
100	Mrad	Amine	Oran
102	Abbés	Amina	Tlemcen
103	Abbés	Eya	Oran
100		Eya	Alger

Order

numCde	amount	ClientNumber
1024	200,00	102
	250,00	170
1204	-310,00	100
1256	100,00	102

Anomaly encountered	Constraint not respected and explanation
In the client table, two clients have the same number 100.	Integrity constraint of the key: the primary key must be unique.
In the Command table, the second command does not have a number.	Integrity constraint of the key: the primary key must not be NULL.
In the Command table, the amount of the 3rd command is negative.	Domain integrity constraint: attribute values must belong to the domain.
In the Command table, the customer number 170 does not exist in the Client table.	Referential integrity constraint: foreign key values are values from the primary key they reference.

4.4. Rules for converting the entity-relationship model to the relational model (MLD)

Converting an Entity-Relationship (ER) model to a Relational Model (MLD) involves applying specific rules to ensure that the data structure is appropriately represented in relational terms. Here are the key rules for this translation process:

4.4.1. Entity Types to Relations

Each entity type in the ER model becomes a relation (table) in the relational model. The attributes of the entity are translated into the attributes (columns) of the relation.

Example: An entity type STUDENT with attributes STUDENT_ID, STUDENT_NAME, and DATE_OF_BIRTH becomes a relation:

STUDENT (STUDENT_ID, STUDENT_NAME, DATE_OF_BIRTH)

4.4.2. Identifier as Primary Key

The identifier (primary key) of each entity in the ER model is designated as the primary key of the corresponding relation in the relational model.

Example: If STUDENT_ID is the identifier for the STUDENT entity, it becomes the primary key:

STUDENT (STUDENT_ID PRIMARY KEY, STUDENT_NAME, DATE_OF_BIRTH)

4.4.3. Relationships with Maximum Cardinality of 1

For relationships where both ends have a maximum cardinality of 1, include the primary key of one of the relations as a foreign key in the other relation.

Example: If STUDENT and ADDRESS both have a maximum cardinality of 1 in their relationship, add ADDRESS_ID (the primary key of ADDRESS) as a foreign key in the STUDENT table:

STUDENT (<u>STUDENT_ID</u>, STUDENT_NAME, <u>ADDRESS_ID</u>)

4.4.4. Relationships with Maximum Cardinality of N and 1

For relationships where one end has a maximum cardinality of N and the other has a maximum cardinality of 1, include the primary key of the relation with a maximum cardinality of N as a foreign key in the relation with a maximum cardinality of 1.

Example: If COURSE (maximum cardinality N) is related to STUDENT (maximum cardinality 1), include STUDENT_ID as a foreign key in the COURSE table:

COURSE (COURSE_ID, STUDENT_ID)

4.4.5. Relationships with Maximum Cardinality of N

For relationships where both ends have a maximum cardinality of N, create a new relation that includes a primary key formed by concatenating the primary keys of the participating relations. Any attributes associated with the relationship are added to this new relation.

Example: For a relationship ENROLLMENT between STUDENT and COURSE, both of which can have multiple instances, create a new relation:

ENROLLMENT (STUDENT_ID, COURSE_ID, GRADE)

Here, STUDENT_ID and COURSE_ID form the composite primary key, and GRADE is an attribute of the relationship.

4.5. Exercises

4.5.1. Exercise 1

Redo the same exercise 1 from Chapter 3 and provide the corresponding relational model.

4.5.2. Exercise 2

Redo the same exercise 2 from Chapter 3 and provide the corresponding relational model.

4.5.3. Exercise 3

Redo the same exercise 3 from Chapter 3 and provide the corresponding relational model.

4.5.4. Exercise 4

Redo the same exercise 4 from Chapter 3 and provide the corresponding relational model.

4.6. Solution of exercises

4.6.1. Solution of exercise 1

Book (Code, Title, Author, Publisher, Date)

Subscriber (Number, last_name, first_name, address, phone)

Borrow (Code, Number, date_L, date_R)

4.6.2. Solution of exercise 2

Client (Number, surname, first_name, phone)

Room (NumberRo, Surface, Sea_view, NumberT)

Type (NumberT, NbPeople)

Reservation (NumberR, Type_board, Number, NumberRo, Period)

4.6.3. Solution of exercise 3

Employee (NSC, Name, Last_name, Birth_Date, Hire_date, Salary, idF, idD)

Function (idF, NameF)

Department (idD, NameD)

Leave (NumL, TypeL, NSC, Date, duration)

4.6.4. Solution of exercise 4

University (NameU, AdrU)

Conference (NameC, Date, NameU)

Presentation (Title, summary)

Speaker (NumS, NameS, First_NameS, AdrS, NameU)

Organizer (NumO, NameO, First_NameO, AdrO,Phone, NameC))

Participant (NumP, NameP, First_NameP, AdrP, NameC)

Participate (NameC, NumP)

Present (NameC, Title, NumS)

CHAPTER 5

SQL LANGUAGE

5.1. Definition

SQL (Structured Query Language) is the standard language used for managing and manipulating relational databases [3]. It enables users to create database objects (like tables and views), perform data manipulation operations (such as inserting, updating, and querying data), and manage data security through access control commands. SQL is integral to leading relational database management systems (RDBMS) such as Oracle, Microsoft SQL Server, MySQL, Access, and IBM DB2. Its versatility and power make it essential for database administrators, developers, and data analysts. As data-driven decision-making becomes increasingly important, proficiency in SQL remains a vital skill in the tech industry.

5.2. Data Definition Language (DDL)

DLL is used to define and modify the database structure or schema. It allows users to create, alter, and delete database objects such as tables, indexes, views, and constraints. Here are the primary DDL commands with example code relevant to Microsoft Access [7].

5.2.1. Creating Database Objects

The CREATE TABLE command is used to create a new table in the database.

Example: Creating a Customers table.

```
CREATE TABLE Customers (
    CustomerID AUTOINCREMENT PRIMARY KEY,
    CustomerName TEXT(100),
    ContactEmail TEXT(100),
    PhoneNumber TEXT(15)
);
```

CustomerID is defined as an AUTOINCREMENT field, which automatically generates a unique number for each record.

CustomerName, ContactEmail, and PhoneNumber are defined as TEXT fields with specified lengths.

5.2.2. Altering Existing Database Objects

The ALTER TABLE command is used to modify an existing table structure. This includes adding or modifying columns or constraints.

Example: Adding a new column DateOfBirth to the Customers table.

```
ALTER TABLE Customers

ADD DateOfBirth DATE;
```

This command adds a new column named DateOfBirth of type DATE to the existing Customers table.

5.2.3. Deleting Database Objects

The DROP TABLE command is used to delete an entire table from the database, including all of its data.

Example: Dropping the Customers table.

```
DROP TABLE Customers;
```

This command removes the Customers table and all the data contained within it from the database.

5.2.4. Creating Indexes

Indexes are created to improve the speed of data retrieval operations. In Access, the CREATE INDEX command is used.

Example: Creating an index on the CustomerName field in the Customers table.

```
CREATE INDEX idx_CustomerName ON Customers(CustomerName);
```

This command creates an index named idx_CustomerName on the CustomerName field to optimize queries that search by customer names.

5.2.5. Defining Constraints

Constraints can be defined while creating or altering tables to enforce data integrity.

Example: Creating an Orders table with a foreign key constraint referencing the Customers table.

```
CREATE TABLE Orders (
    OrderID AUTOINCREMENT PRIMARY KEY,
    CustomerID LONG,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
```

);

This command creates an Orders table with a foreign key constraint that ensures any CustomerID entered in the Orders table corresponds to an existing CustomerID in the Customers table.

5.3. Data Manipulation Language (DML)

DML is a subset of SQL used for managing and manipulating data stored within a relational database. DML encompasses a variety of commands that allow users to perform operations such as inserting, updating, deleting, and retrieving data g[8], [9].

5.3.1. Inserting Data

The INSERT command is used to add new records to a table. Users can insert a single row or multiple rows at once. To add a tuple to a table, the following steps are followed:

INSERT INTO table_name

VALUES (attribute_value1, attribute_value2, ...) ;

Example: Inserting a single record into the Customers table.

INSERT INTO Customers (CustomerName, ContactEmail, PhoneNumber)

VALUES ('Alaoui Amina', 'alaoui.amina@gmail.com', '0771717173');

Example: Inserting multiple records.

INSERT INTO Customers (CustomerName, ContactEmail, PhoneNumber)

VALUES

('Alaoui Amina', 'alaoui.amina@gmail.com', '0771717173'),

('Madani Ahmed', 'madani.ahmed@gmail.com', '0555110789');

5.3.2. Updating Data

The UPDATE command modifies existing records in a table based on specified conditions.

UPDATE table_name

SET attribute1 = value1, attribute2 = value2, ...

WHERE condition

Example: Updating the email address of a customer in the Customers table.

```
UPDATE Customers
SET ContactEmail = 'alaoui.newemail@gmail.com'
WHERE CustomerName = 'Alaoui Amina';
```

5.3.3. Deleting Data

The DELETE command removes records from a table based on specified criteria.

DELETE FROM table_name

WHERE condition

Example: Deleting a customer from the Customers table.

```
DELETE FROM Customers
WHERE CustomerName = 'Madani Ahmed';
```

5.3.4. Retrieving Data

The SELECT command is used to retrieve data from one or more tables. Users can specify which columns to retrieve and apply conditions to filter results.

Displays all attributes of a table.

**SELECT *

FROM table_name;

Some attributes:

SELECT attribut1, attribut2, ...

FROM table_name;

Example: Retrieving all records from the Customers table.

```
SELECT * FROM Customers;
```

Example: Retrieving specific columns and filtering results.

```
SELECT CustomerName, ContactEmail
FROM Customers
WHERE PhoneNumber LIKE '555-%';
```

5.3.5. Aggregating Data

DML also allows users to perform aggregate functions, such as counting records, calculating sums, or finding averages.

In Microsoft Access, functions are utilized to perform various data manipulations and calculations within queries, forms, and reports. Common types of functions include aggregate functions like SUM, AVG, and COUNT, which compute values across multiple records. String functions, such as LEFT and MID, manipulate text data, while date and time functions like NOW and DATEDIFF work with date values. Mathematical functions like ABS and ROUND perform calculations on numeric data. Additionally, conditional functions like IIf and Switch allow users to evaluate conditions and categorize data accordingly, enhancing data analysis and reporting capabilities.

Example: Counting the number of customers in the Customers table.

SELECT COUNT(*) AS TotalCustomers FROM Customers;

Example: Finding the average salary from an Employees table.

SELECT AVG(Salary) AS AverageSalary FROM Employees;

Here's a table summarizing the functions in Microsoft Access, categorized by type:

Function Type	Function	Description	Example
Aggregate Functions	SUM	Calculates the total of a numeric field.	SELECT SUM(Salary) FROM Employees;
	AVG	Computes the average of a numeric field.	SELECT AVG(Salary) FROM Employees;
	COUNT	Counts the number of records.	SELECT COUNT(*) FROM Customers;
	MAX	Returns the maximum value in a set.	SELECT MAX(Salary) FROM Employees;
	MIN	Returns the minimum value in a set.	SELECT MIN(Salary) FROM Employees;
	LEFT	Returns a specified number of	SELECT LEFT(CustomerName, 5) FROM Customers;

String Functions		characters from the left side.	
	RIGHT	Returns a specified number of characters from the right side.	SELECT RIGHT(CustomerName, 3) FROM Customers;
	MID	Returns a substring starting at a specified position.	SELECT MID(CustomerName, 2, 3) FROM Customers;
	LEN	Returns the length of a string.	SELECT LEN(CustomerName) FROM Customers;
Date Functions	NOW	Returns the current date and time.	SELECT NOW();
	DATE	Returns the current date.	SELECT DATE();
	DATEDIFF	Calculates the difference between two dates.	SELECT DATEDIFF('d', HireDate, NOW()) FROM Employees;
Mathematical Functions	ABS	Returns the absolute value of a number.	SELECT ABS(-100);
	ROUND	Rounds a number to a specified number of decimal places.	SELECT ROUND(Salary, 2) FROM Employees;
Conditional Functions	IIf	Returns one value if a condition is true and another if false.	SELECT IIf(Salary > 50000, 'High', 'Low') FROM Employees;
	Switch	Evaluates a series of expressions and returns the first true value.	SELECT Switch(Salary < 30000, 'Low', Salary >= 30000, 'Medium') FROM Employees;

5.3.6. Sorting and Grouping Data

Users can sort retrieved data and group results based on specific criteria.

Sorting data enables users to arrange records in a specified order, either ascending or descending. The ORDER BY clause is used in SQL queries to achieve this.

- Ascending Order: Default order where values are sorted from the smallest to the largest (e.g., A to Z, 0 to 9).
- Descending Order: Values are sorted from the largest to the smallest (e.g., Z to A, 9 to 0).

Example: Sorting customers by their last names in ascending order.

```
SELECT * FROM Customers
ORDER BY CustomerName ASC;    ASC is optional as it is the default
```

Example: Sorting employees by their salary in descending order.

```
SELECT * FROM Employees
ORDER BY Salary DESC;
```

Grouping data allows users to aggregate records based on shared attributes, making it easier to analyze patterns and trends. The GROUP BY clause is used in combination with aggregate functions to summarize data.

Example: Grouping data and counting customers by city.

```
SELECT City, COUNT(*) AS NumberOfCustomers
FROM Customers
GROUP BY City;
```

5.3.7. Using HAVING Clause with Grouping

When filtering grouped results, the HAVING clause is used in conjunction with GROUP BY. It allows users to specify conditions on aggregate values.

Example: Finding cities with more than five customers.

```
SELECT City, COUNT(*) AS NumberOfCustomers
FROM Customers
GROUP BY City
HAVING COUNT(*) > 5;
```

5.3.8. Joins in SQL

In SQL, a join is used to combine rows from two or more tables based on a related column between them. When multiple tables are specified in the FROM clause without a join condition, a Cartesian product (or cross join) is produced, which pairs every row of one table with every row of another. However, this often results in an impractically large dataset, so joins are used to filter the results meaningfully.

5.3.8.1. Cartesian Product

The Cartesian product occurs when there are no conditions specified to limit the relationships between the tables. Each row from the first table is combined with every row from the second table, leading to a potentially massive output.

Example:

```
SELECT *
FROM TableA, TableB;  -- Produces a Cartesian product of TableA and TableB
```

5.3.8.2. Types of Joins

There are several types of joins that allow users to retrieve specific related data from multiple tables:

- **Inner Join**: Returns only the rows that have matching values in both tables.
- **Left Join (or Left Outer Join)**: Returns all rows from the left table and the matched rows from the right table. If there is no match, NULL values are returned for columns from the right table.
- **Right Join (or Right Outer Join)**: Returns all rows from the right table and the matched rows from the left table. If there is no match, NULL values are returned for columns from the left table.
- **Full Join (or Full Outer Join)**: Returns all rows when there is a match in either left or right table. Rows without matches will have NULL values in columns from the table without a match.

Example of Inner Join:

```
SELECT A.*, B.*
FROM TableA A
```

INNER JOIN TableB B ON A.commonColumn = B.commonColumn; -- Combines rows where the common Column matches

5.3.8.3. Self-Join

A self-join is a specific type of join where a table is joined with itself. This can be useful when you need to compare rows within the same table. To perform a self-join, the table must be aliased with different names to differentiate between the two instances.

Example: Consider an Employees table where each employee has a ManagerID that references another employee in the same table.

SELECT E1.EmployeeName AS Employee, E2.EmployeeName AS Manager

FROM Employees E1

INNER JOIN Employees E2 ON E1.ManagerID = E2.EmployeeID; -- Combines employees with their respective managers

In this example: E1 and E2 are aliases for the Employees table.

The join condition matches the ManagerID from E1 with the EmployeeID from E2, effectively pairing each employee with their manager.

5.4. Exercise

Exercise: Given the following relational schema:

Client (ClientId, Name, City)

Product (ProductId, ProductName, Brand, Price, StockQuantity)

Order (ClientId, ProductId, Date, Quantity)

Questions: Provide the SQL queries allowing to:

1) Add the client (510, 'Madani', 'Oran') to the Client table.
2) Increase the price of all products by 20%.
3) Delete clients residing in Oran.
4) Provide all information about the clients.
5) Provide the names, brands, and prices of the products.
6) List the various brands of products.

7) Add to a table "temp" with the same schema as the Client table, all clients who live in Oran.

8) List the names of HP brand products.

9) List the products with prices ranging from 5000 DA to 12000 DA.

10) List the products of HP and Apple brands.

11) List the products with unknown prices.

12) List the IHP brand products with a price lower than 12000 DA.

13) List the products by sorting them by brands and within a brand by descending price.

14) Provide the references of HP brand products or those purchased by client No 123.

15) Provide the total number of clients.

16) Provide the references and names of the products sold.

17) Provide the names of the clients who have purchased the product named 'keyboard'.

18) List the names of clients from the same city as Absar.

19) List the names of the clients who have purchased product p1.

20) Provide the total number of mouse sold.

5. 5. Solution of the exercise

1) Add the client (510, 'Madani', 'Oran') to the Client table.

```
INSERT INTO Client (ClientId, Name, City) VALUES (510, 'Madani', 'Oran');
```

2) Increase the price of all products by 20%.

```
UPDATE Product SET Price = Price * 1.20;
```

3) Delete clients residing in Oran.

```
DELETE FROM Client WHERE City = 'Oran';
```

4) Provide all information about the clients.

```
SELECT * FROM Client;
```

5) Provide the names, brands, and prices of the products.

```
SELECT ProductName, Brand, Price FROM Product;
```

6) List the various brands of products.

```
SELECT DISTINCT Brand FROM Product;
```

7) Add to a table "temp" with the same schema as the Client table, all clients who live in Oran.

```
INSERT INTO temp (ClientId, Name, City)
SELECT ClientId, Name, City FROM Client WHERE City = 'Oran';
```

8) List the names of HP brand products.

```
SELECT ProductName FROM Product WHERE Brand = 'HP';
```

9) List the products with prices ranging from 5000 DA to 12000 DA.

```
SELECT * FROM Product WHERE Price BETWEEN 5000 AND 12000;
```

10) List the products of HP and Apple brands.

```
SELECT * FROM Product WHERE Brand IN ('HP', 'Apple');
```

11) List the products with unknown prices.

```
SELECT * FROM Product WHERE Price IS NULL;
```

12) List the IHP brand products with a price lower than 12000 DA.

```
SELECT * FROM Product WHERE Brand = 'IHP' AND Price < 12000;
```

13) List the products by sorting them by brands and within a brand by descending price.

```
SELECT * FROM Product ORDER BY Brand, Price DESC;
```

14) Provide the references of HP brand products or those purchased by client No 123.

```
SELECT DISTINCT ProductId
FROM Product
WHERE Brand = 'HP'
UNION
SELECT DISTINCT ProductId
FROM Order
WHERE ClientId = 123;
```

15) Provide the total number of clients.

```
SELECT COUNT(*) AS TotalClients FROM Client;
```

16) Provide the references and names of the products sold.

```
SELECT DISTINCT Product.ProductId, Product.ProductName
FROM Product
WHERE  Product.ProductId = Order.ProductId;
```

17) Provide the names of the clients who have purchased the product named 'keyboard'.

```
SELECT DISTINCT Client.Name
FROM Client
WHERE Product.ProductName = 'keyboard' and Client.ClientId = Order.ClientId
and Order.ProductId = Product.ProductId;
```

18) List the names of clients from the same city as Absar.

```
SELECT Name
FROM Client
WHERE City = (SELECT City FROM Client WHERE Name = 'Absar');
```

19) List the names of the clients who have purchased product p1.

```
SELECT DISTINCT Client.Name
FROM Client
WHERE Client.ClientId = Order.ClientId and Order.ProductId = 'p1';
```

20) Provide the total number of mouse sold.

```
SELECT SUM(Order.Quantity) AS TotalMouseSold
FROM Order
WHERE Order.ProductId = Product.ProductId and Product.ProductName = 'mouse';
```

CHAPTER 6

APPLICATION WITH ACCESS

Microsoft Access is a relational database management system (RDBMS) that combines a graphical user interface with powerful database tools. It allows users to create, manage, and analyze data easily using tables, queries, forms, and reports **[**10].

6.1 Database Creation

We will create the "Company" database to store essential tables for managing client, product, and order information. This database will organize and centralize data for streamlined business operations.

Click on "New Database" and enter "Company.accdb" as the database name, figure 6.1.

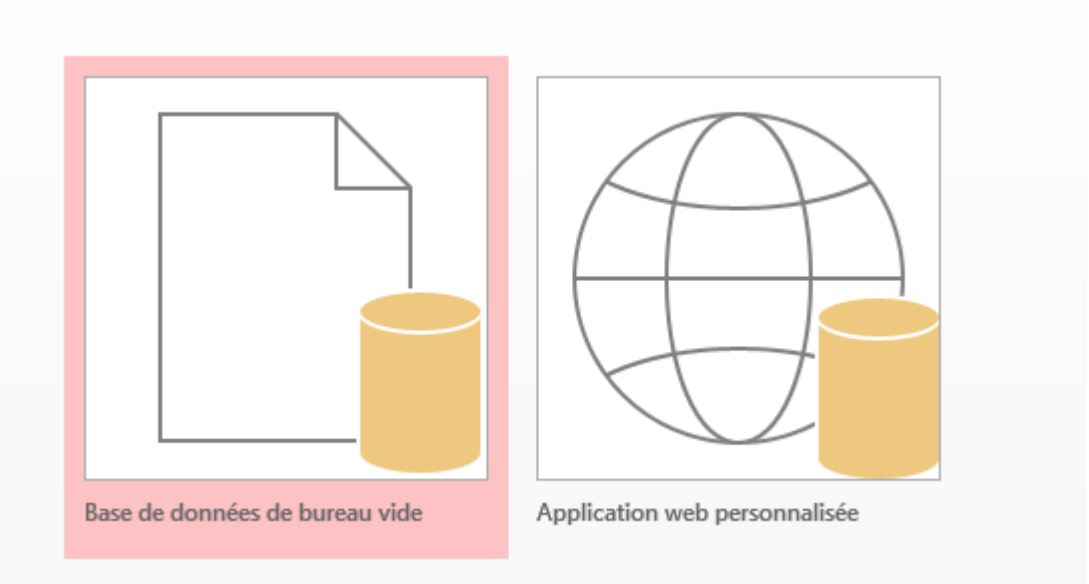


Figure 6.1. Creation of new DB

6.2. Table Creation

To create a table, click on the Create menu, then select Table, and open the table in Design View.

6.2.1. Data Types

Each field can contain data of different types like text, date, numeric, …, see figure 6.2.

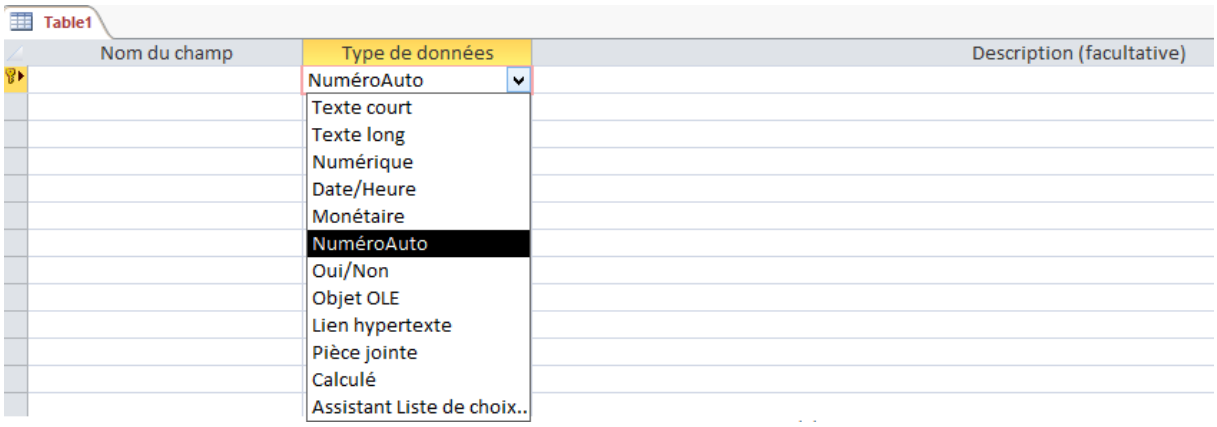


Figure 6.2. The different types of data.

6.2.2. Field properties

As you can see, each field has properties that depend on its data type, see figure 6.3. These properties determine how the data it contains will be displayed or how data will be entered into this field. These properties are displayed in the "General" tab at the bottom of the window. The "General" tab shows each field property and, next to it, in editable fields, the value of that property.

Figure 6.3. Field properties.

Field Properties Explained:

Field Size: Maximum number of characters that can be entered.

Default Value: Text that will be contained by default in this field.

Validation Rule: Expression to validate input; for example, <> "machin" will prevent the entry of the word "machin" in this field.

Validation Message: Error message displayed if the entered expression is invalid; for instance, you could set this to "Entry of 'machin' is prohibited."

Allow Null: If set to "Yes," a value must be entered in this field.

Allow Zero-Length String: The empty string "" can be used when there is nothing to enter in a required field. If set to "No," text must be entered in this field.

Indexed: Determines whether the field is indexed with or without duplicates. If the index does not allow duplicates, it will not be possible to enter the same value twice for this field in the table.

Format: Defines how the content of the field will be displayed (details provided later).

Input Mask: Requires the input to conform to a specific format (e.g., a phone number).

6.2.3. Choice Lists

Choice lists are a way to simplify data entry in a table by allowing the user to click on an item from a provided list, see figure 6.4. The field in the table will be filled with the selected item from the list.

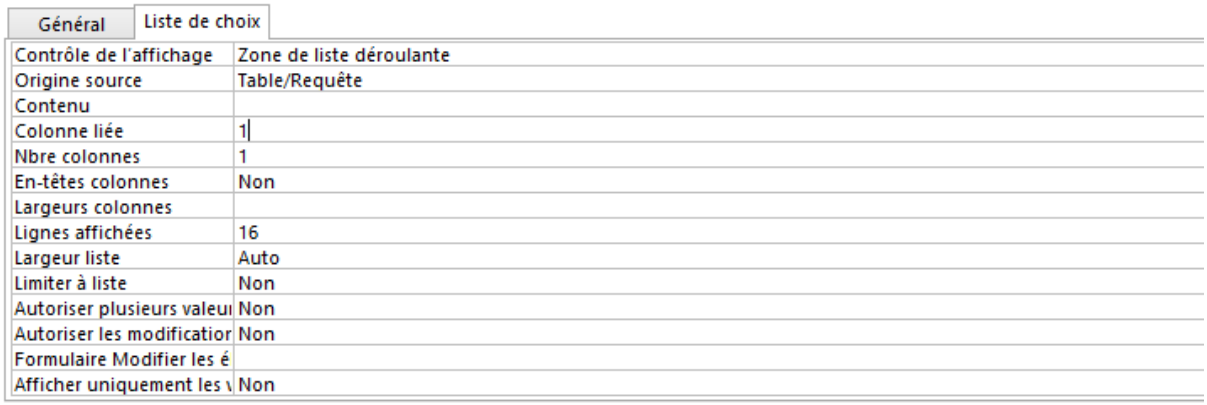


Figure 6.4. Choice lists.

6.3. Entering Records in Datasheet View in Access

To enter records in Microsoft Access in Datasheet View, see figure 6.5, follow these steps:

1. Open the Table: Navigate to the Navigation Pane, find the table where you want to enter records (e.g., Client), and double-click it or right-click and select Open to view it in Datasheet View.

2. Enter Data: In Datasheet View, locate the first empty row at the bottom of the table. Click on the cells in each field and start entering your data. Use the Tab key to move to the next field or use the arrow keys to navigate between cells.
3. Save the Record: After entering the data for each record, press Enter to save the entry. This will add the new record to the table, and you can continue entering more records in the same way.
4. Close the Table: Once you've finished entering your records, you can close the table by clicking the X in the upper right corner of the window or by selecting Close from the File menu.

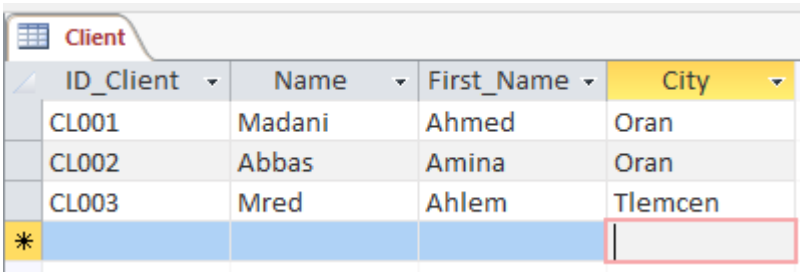


Figure 6.4. Entering records in Datasheet view mode.

6.4. Create Request

6.4.1. Query design mode

To create a query (request) in Microsoft Access, see figure 6.5, follow these steps:

1. Go to the Create Tab: In the main Access window, select the Create tab on the Ribbon.
2. Choose Query Design: Click on Query Design in the Queries group. This will open a new query in Design View and show the Show Table dialog.
3. Add Tables: In the Show Table dialog, select the tables you want to include in your query (e.g., Client, Product, Order) and click Add. Close the dialog once you've added the necessary tables.
4. Select Fields: In Design View, double-click on the fields you want to include in your query, or drag them from the table to the grid at the bottom. For example, to view client names and cities, select the Name and City fields from the Client table.

5. Set Criteria (Optional): To filter results, enter criteria in the Criteria row of a field. For example, to list only clients from "Oran," type "Oran" in the Criteria row under the City field.
6. Run the Query: Click on Run (represented by a red exclamation mark) in the Design tab to view the results of your query.
7. Save the Query: To save the query, click Save in the Quick Access Toolbar, give the query a name, and click OK.

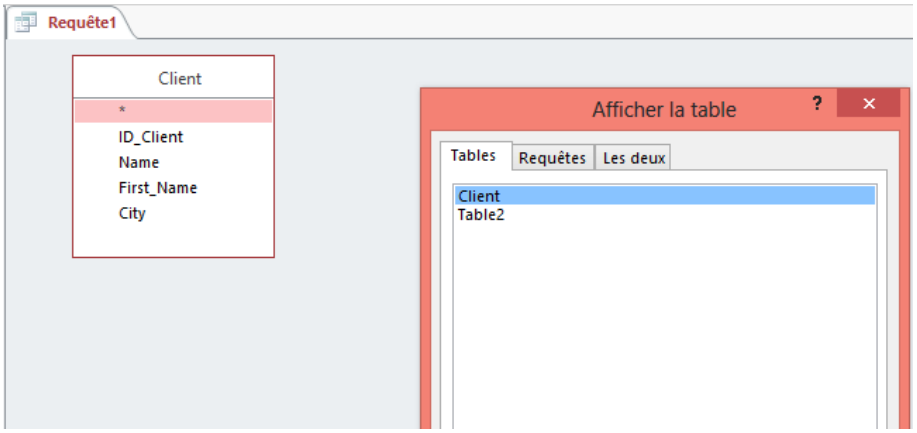


Figure 6.5. Creating a query in Design mode.

6.4.2. SQL mode

To create a query using SQL code directly in Access, see figure 6.6. , you can follow these steps:

1. Go to the Create Tab: Open Microsoft Access, and on the top Ribbon, click on the Create tab.
2. Choose Query Design: In the Create tab, click on Query Design in the Queries group.
3. Switch to SQL View: In the query design window, click on SQL View. You'll find this option in the View dropdown menu at the top-left of the screen, or you can select it directly in the Design tab.
4. Write the SQL Code: Enter your SQL code directly into the SQL editor.

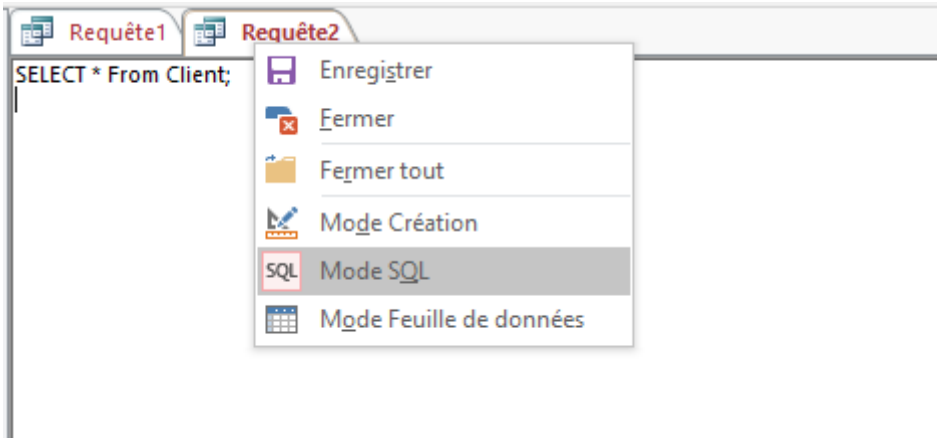


Figure 6.6. Creating a query in SQL mode.

Examples:

Select All Records from a Table:

SELECT * FROM Client;

Select Specific Fields:

SELECT Name, City FROM Client;

Select with a Condition:

SELECT Name, City

FROM Client

WHERE City = 'Oran';

Join Tables:

SELECT Client.Name, Product.ProductName, Order.Quantity

FROM Client, Order, Product

Client.ClientId = Order.ClientId and Order.ProductId = Product.ProductId;

Insert Data:

INSERT INTO Client (ClientId, Name, City)

VALUES (510, 'Madani', 'Oran');

Update Data:

UPDATE Product

SET Price = Price * 1.20;

5. Run the Query: Click on Run (red exclamation mark in the Design tab) to execute your SQL code.

6. Save the Query: If you want to save this SQL query for future use, click on the Save icon, give the query a name, and click OK.

6.5. Macro

6.5.1. Definition of a Macro

In Microsoft Access, a macro is a tool that automates tasks and adds functionality to database. In fact, a macro is a set of predefined actions or instructions that can be executed as a single command to perform specific tasks, such as opening forms, running queries, or displaying messages.

6.5.2. Key features of macros

- Automation: macros reduce repetitive tasks by automating them with a single click or event.
- No programming required: macros do not require coding knowledge, making them accessible for beginners.
- User-Friendly: Macros are created using a graphical interface, where it's possible to select actions and define parameters.
- Conditional logic: macros can include conditions to perform actions only when specific criteria are met.

6.5.3. Benefits of using macros

- Simplifies complex tasks: can create workflows without writing code.
- Error reduction: automates repetitive tasks, minimizing human error.
- Increased productivity: saves time by automating frequently used operations.

6.5.4. Steps to create macros in Access

To create a macro in Access, it will be necessary to follow these steps, see figure 6.7:

Step 1: Open the Macro Builder

1. Open your Access database.
2. Go to the Create tab in the ribbon.
3. In the Macros & Code group, click on Macro.

- This opens the Macro Builder, where you can define the actions your macro will perform.

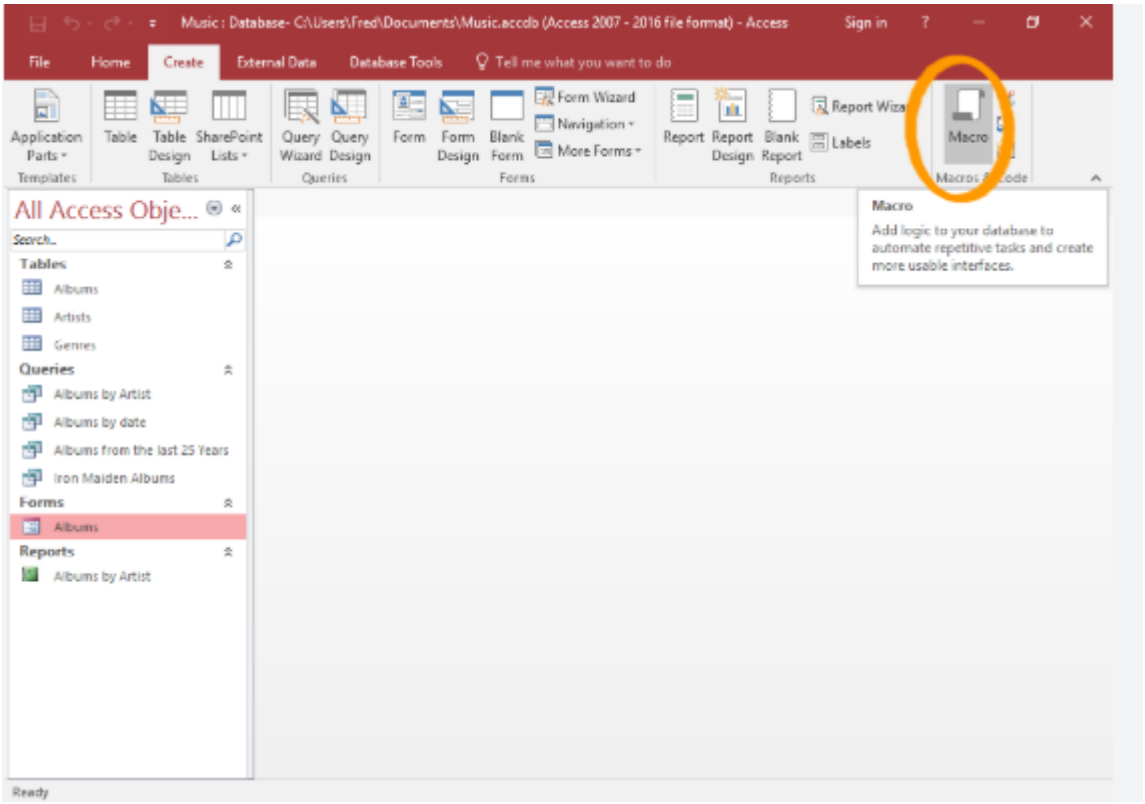


Figure 6.7. Creating Macro.

Step 2: Add Macro Actions

1. In the Macro Builder, click on the first row under the Action column.
2. Click the dropdown arrow to see the list of available actions.
3. Select the action you want the macro to perform, see figure 6.8. For example:
    - OpenForm: To open a form.
    - OpenTable: To open a table.
    - OpenQuery: To run a query.
    - CloseWindow: To close a window.
    - MessageBox: To display a message to the user.

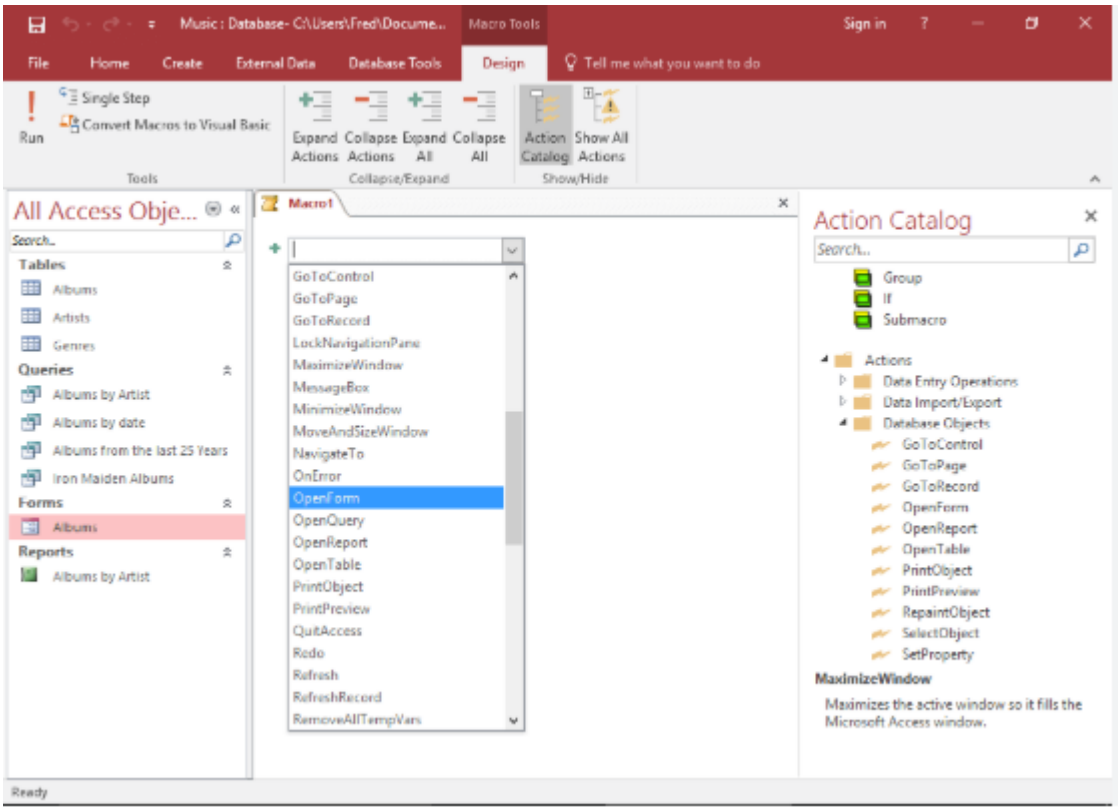


Figure 6.8. Macro actions in Access

Example: Add an OpenForm Action

- Select OpenForm from the action list.
- Configure its arguments in the lower pane:
    - Form Name: Select the form you want to open.
    - View: Choose the view (e.g., Form).
    - Filter Name: Leave blank unless applying a filter.
    - Where Condition: Leave blank unless applying a specific condition.
    - Data Mode: Choose Edit, Add, or Read Only (default is Edit).
    - Window Mode: Choose Normal, Minimized, or Dialog (default is Normal).

Step 3: Add Additional Actions

1. Add more rows for additional actions. For each row:
    - Select the action type from the dropdown.
    - Configure its arguments as needed.

Example Sequence of Actions:

- OpenTable: Open a table in datasheet view.
- MessageBox: Display a message to the user.
- CloseWindow: Close a form or table after the user acknowledges the message.

Step 4: Save the Macro

1. Click the Save button (disk icon) in the toolbar.
2. Give your macro a meaningful name, e.g., OpenFormMacro.
3. Close the Macro Builder.

Step 5: Run the Macro

1. In the Navigation Pane, locate your macro under the Macros section.
2. Double-click the macro to execute it and verify its actions.

Step 6: Assign the Macro to a Button (Optional)

1. Open a form in Design View.
2. Add a button to the form:
    - Go to the Design tab, click the Button control, and place it on the form.
3. When the Command Button Wizard appears:
    - Select Miscellaneous > Run Macro.
    - Choose your macro from the list.
    - Finish the wizard and assign a label to the button (e.g., "Run Macro").
4. Save the form and switch to Form View to test the button.

Step 7: Set the Macro to Run Automatically

1. To run the macro when the database opens:
    - Rename your macro to AutoExec.
    - The AutoExec macro will automatically run whenever the database is opened.

Common Macro Actions and Their Use

Action	Description
OpenForm	Opens a specified form.
OpenTable	Opens a table in datasheet view.
OpenQuery	Runs a query.
RunMacro	Runs another macro from within the current macro.
MessageBox	Displays a custom message box to the user.
CloseWindow	Closes a specific window (form, table, etc.).

6.6. Form in Microsoft Access

6.6.1. Definition of a Form

In Microsoft Access, a form is a database object that provides a user-friendly interface for viewing, entering, and managing data stored in tables. Forms are typically designed to make interacting with the database easier and more intuitive by organizing fields, adding controls, and customizing the layout.

6.6.2. Key Features of Forms

1. Data Entry: Forms are commonly used for entering and updating data in tables.
2. Data Display: Forms can display data from one or multiple related tables in a visually organized format.
3. Customization: You can add buttons, drop-down menus, search fields, and other controls to make the form interactive.
4. Validation: Forms can include rules to validate data before saving it to a table.

6.6.3. Uses of Forms

1. Simplifying Data Input: Forms make it easier for users to input data compared to directly working with tables.
2. Creating Menus: Forms can be used as navigation menus for the database (e.g., main menu).

3. Filtering and Sorting: Forms can include filters or search functionalities to display only relevant data.
4. Enhancing User Experience: By customizing the design, forms can make the database more intuitive and visually appealing.

6.6.4. Steps to create Form in Access

Here is a step-by-step guide to create a form in Microsoft Access:

Step 1: Open Your Database

1. Open the Access database where you want to create the form.

Step 2: Use the Form Wizard or Create a Blank Form

There are two main ways to create a form in Access:

Option 1: Use the Form Wizard

1. Go to the Create tab on the ribbon, see figure 6.9.
2. In the Forms group, click on Form Wizard.
3. Follow the steps in the wizard:
    - **Step 1**: Select the table or query from which the form will pull data.
    - **Step 2**: Choose the fields you want to include in the form.
        - Use the >> button to include all fields or > to add selected fields.
    - **Step 3**: Choose a layout for your form:
        - **Columnar**: Fields are displayed in a single column.
        - **Tabular**: Fields are displayed in a table-like format.
        - **Datasheet**: Similar to a table view.
        - **Justified**: Fields are spread out across the form.
    - **Step 4**: Give your form a name and click Finish.
4. Access will create the form and display it in Form View.

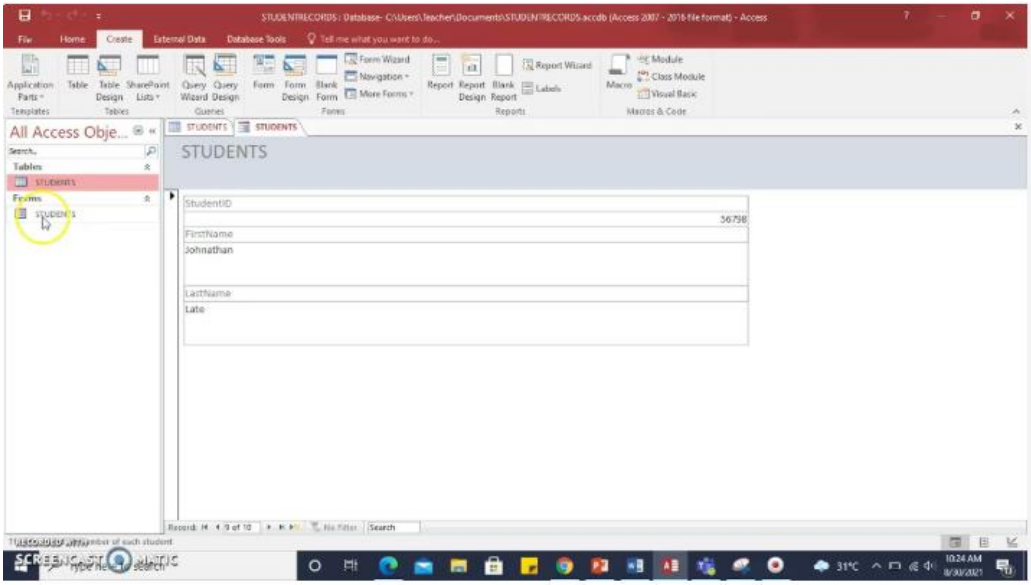


Figure 6.9. Create Form in Access

Option 2: Create a Blank Form (For Customization)

1. Go to the Create tab on the ribbon, see figure 6.10.
2. In the Forms group, click Blank Form.
3. The form opens in Layout View or Design View.
4. Use the Field List Pane (accessible via the Add Existing Fields button in the Design tab) to drag and drop fields from a table or query onto the form.
5. Customize the form layout by adding controls, labels, or formatting as needed.

Step 3: Add Form Controls

1. In Design View or Layout View, use the Design tab to add controls such as:
    - **Text Boxes**: For data entry and display.
    - **Combo Boxes**: For drop-down lists.
    - **Buttons**: For actions like saving, deleting, or opening other forms.
    - **Labels**: For adding titles or descriptions.
2. Arrange the fields and controls to make the form user-friendly.

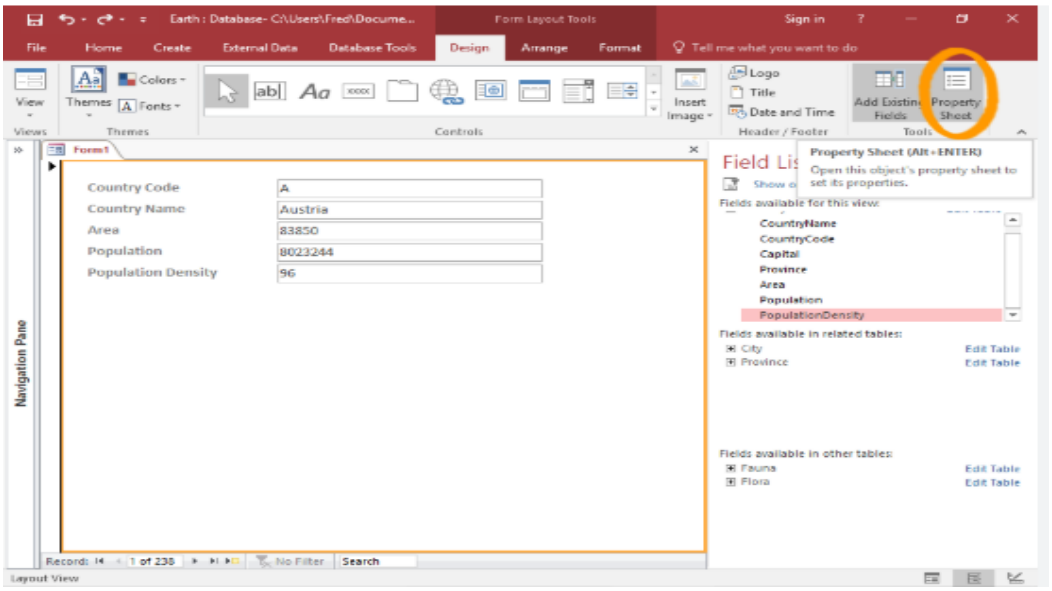


Figure 6. 10. Create Form in Access

Step 4: Apply Design and Formatting

1. Switch to Design View to adjust the layout and appearance of the form.
2. Use the Format tab to:
    - Change fonts, colors, and themes.
    - Add a logo or background image.
    - Align and resize controls for a polished look.

Step 5: Save the Form

1. Click the Save button (disk icon) in the toolbar.
2. Give the form a meaningful name, such as BooksForm or CustomerEntryForm.

Step 6: Test the Form

1. Switch to Form View (available in the View dropdown in the ribbon).
2. Interact with the form to ensure it works as expected:
    - Enter data and confirm it updates the corresponding table.
    - Test buttons or other controls if added.

Step 7: Add the Form to Navigation (Optional)

1. If you're building a menu or navigation form, you can link this form to the main menu.
2. Use a button in a menu form or set this form as the default startup form in File > Options > Current Database

6.7. Report in Microsoft Access

6.7.1. Definition of a Report

In Microsoft Access, a report is a database object used to format, summarize, and present data in a structured and printable format. Reports allow users to generate professional-looking documents based on the data stored in tables or queries.

6.7.2. Key Features of Reports

1. **Data Presentation:** Reports organize and display data in a visually appealing and structured manner.
2. **Summarization:** Reports can group data and calculate totals, averages, counts, and other aggregate values.
3. **Customization:** Users can add headers, footers, images, and formatting to enhance readability.
4. **Filtering and Sorting:** Reports can be filtered to display only relevant records and sorted in a desired order.
5. **Printing and Exporting:** Reports can be printed, saved as PDFs, or exported to other formats for sharing.

6.7.3. Uses of Reports

1. **Generating Summaries:** Reports provide an overview of data, such as sales totals, customer orders, or inventory levels.
2. **Creating Printable Documents:** Reports can generate invoices, receipts, certificates, or other official documents.
3. **Analyzing Data Trends:** Reports can display trends over time through charts, graphs, or grouped data views.
4. **Enhancing Decision-Making:** Well-structured reports help businesses and organizations make informed decisions.

6.7.4. Steps to Create a Report in Access

Step 1: Open Your Database

1. Open the Access database containing the data for your report.

Step 2: Use the Report Wizard or Create a Blank Report

There are two main ways to create a report in Access:

Option 1: Use the Report Wizard (Recommended for Beginners)

1. Go to the Create tab on the ribbon, see figure 6.11.
2. In the Reports group, click Report Wizard.
3. Follow the wizard steps:
    - **Step 1:** Select the table or query as the data source.
    - **Step 2:** Choose the fields to include in the report.
    - **Step 3:** Apply grouping levels if needed (e.g., group by category or region).
    - **Step 4:** Select a sorting order for records.
    - **Step 5:** Choose a report layout (e.g., tabular, columnar, justified).
    - **Step 6:** Name the report and click Finish.
4. Access generates the report and displays it in Print Preview or Report View.

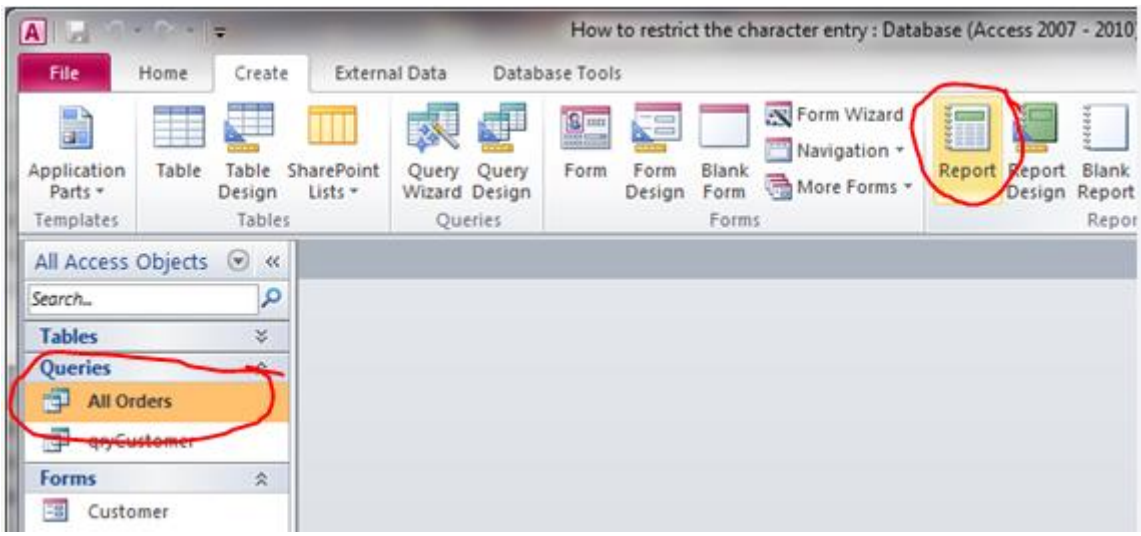


Figure 6.11. Create Report in Access

Option 2: Create a Blank Report (For Full Customization)

1. Go to the Create tab on the ribbon.
2. Click Blank Report in the Reports group.
3. The report opens in Layout View or Design View.
4. Use the Field List Pane (via the Add Existing Fields button) to drag and drop fields onto the report.
5. Customize the report layout by adding text boxes, labels, and formatting elements.

Step 3: Add Report Controls

1. In Design View, use the Design tab to insert elements, see figure 6.12 such as:
    - **Text Boxes:** To display individual data fields.
    - **Labels:** To add titles, descriptions, or captions.
    - **Group Headers & Footers:** To organize grouped data.
    - **Images & Logos:** For branding and visual enhancement.
    - **Charts & Graphs:** To visually represent data trends.

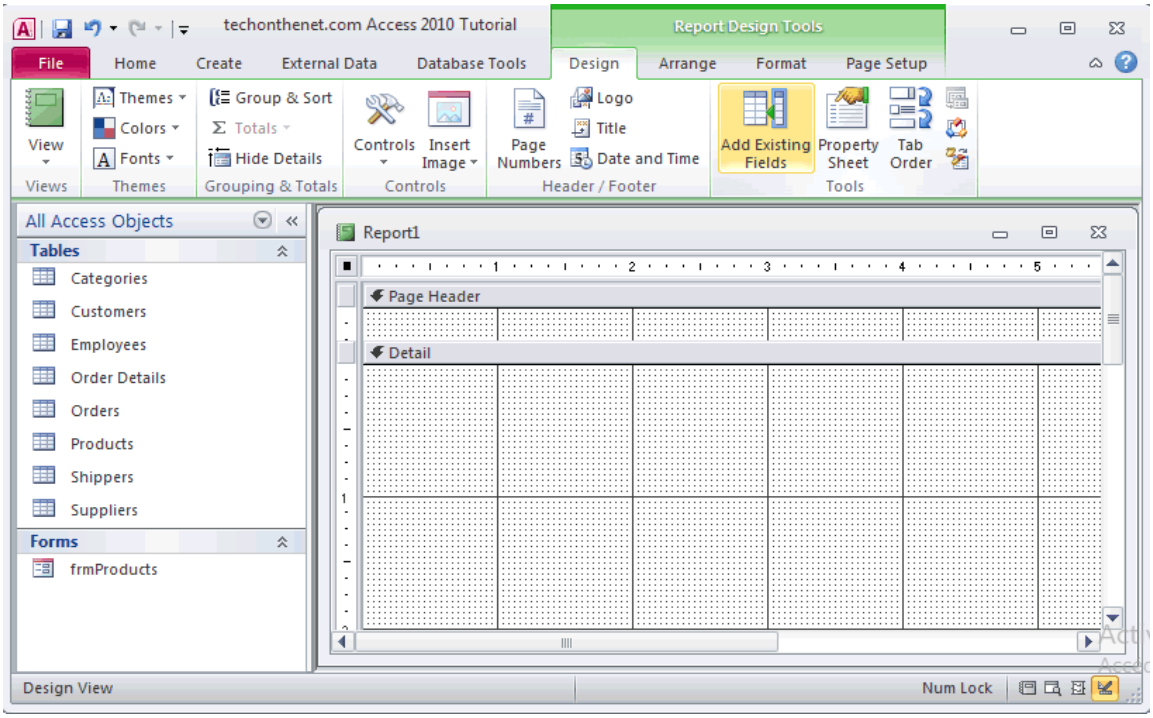


Figure 6.12. Report design view

Step 4: Apply Formatting and Layout Adjustments

1. Switch to Design View to fine-tune the report's appearance.
2. Use the Format tab to:
    - Adjust fonts, colors, and themes.
    - Resize and align controls for a polished look.
    - Insert headers and footers for additional details like dates or page numbers.

Step 5: Save the Report

1. Click the Save button (disk icon) in the toolbar.
2. Provide a meaningful name, such as SalesReport or CustomerInvoices.

Step 6: Preview and Test the Report

1. Switch to Print Preview to check how the report will look when printed.
2. Verify that the data appears correctly and formatting is appropriate.

Step 7: Print or Export the Report

1. To print the report, click File > Print and select your printer.
2. To export the report, go to File > Save As, then choose PDF, Excel, or another format.

6.8. Case study

We propose to represent the information system of a library:

The library records each reader to whom it assigns a reader number. It collects their name, phone number, email, and city. The reader may optionally be a member of an affiliated society. In that case, the identification of this society is recorded. A reader can borrow multiple books each day. For each loan, a "latest return date" is associated. A reader belongs to a reader type. This type determines whether they have access to certain book categories. The loan duration depends on the book category and reader type. It is the same for all books in a given category borrowed by any reader of a given type. A book is characterized by its inventory number. It is necessary to know its title, category, author's name, publisher, and the number of available copies. The edition, if it exists, is also to be known. A book category is identified by a number and has a label. The same applies to the reader type.

1) Creation of tables

a) Create a new database, saved in a file named "Library.accdb".

b) Create the tables associated with the relational model designed in the database "Library.accdb".

2) Simple input validation control

a) We want to capture that the numbers in a phone number and that the dashes are added automatically. For this, Define input masks for the field "Tel".

b) There are certain fields that cannot have a negative value: Number of copies (modify Valid if and Message if error).

c) Define for the field Name the requirement to be entered (Null not accepted).

d) Create an editable list for the City field: Oran, Tlemcen, Mostaganem, Algiers.

e) The vast majority of students reside in Oran.

3) Relationships between tables

a) Define the relationships between tables by enabling referential integrity.

b) Fill these tables with 4 tuples each.

4) Querying the database

a) What is the telephone list of all readers? Only the name and phone number are displayed.

b) We would like to obtain the list of readers. This list is sorted alphabetically by cities and in descending order by names.

c) We want to obtain all information about readers residing in ORAN.

d) Create a query that deletes all readers who live in Tlemcen.

e) List the titles of books in the "Computer Science" category.

f) Provide the number of copies of books with the title "Access".

g) What is the total number of books?

h) Among all the books, which books have the maximum and minimum number of copies? Then, calculate the average of these two values.

i) Display the list of readers sorting them by city (in descending order), and within cities in ascending order by name.

5) Forms

a) Using the wizard, we would like to create a simple data entry screen that facilitates book entry.

b) Create an "Exit" button that allows you to close this form.

6) Reports: Create a report containing the list of books by category.

7) Macros: Create a macro that opens a table and a form.

8) Interface Create a general menu of your choice for this application.

6.9. Solution if the case study

1) Creation of Tables

a) Create a new database:

Open Microsoft Access.

Select File > New.

Name the database as Library.accdb and save it.

b) Create the tables based on the library's requirements:

Readers Table: Fields should include ReaderID, Name, PhoneNumber, Email, City, SocietyID, and ReaderTypeID.

To create a Readers table in Microsoft Access: Open Access and Navigate to Table Design:

Go to the Create tab and select Table.

Switch to Design View to define fields and set their properties.

Define Fields and Data Types: Add the following fields:

- ReaderID (Primary Key) – AutoNumber.
- Name – Short Text.
- PhoneNumber – Short Text.
- Email – Short Text.

- City – Short Text.
- SocietyID – Number (if related to another table like Society).
- ReaderTypeID – Number (if related to another table like ReaderType).

Save the Table:

Click Save, name the table as "Readers," and ensure the primary key (ReaderID) is set.

The table is now ready for data entry or to establish relationships with other tables such as ReaderType for a relational database design.

Books Table: Fields should include InventoryNumber, Title, CategoryID, Author, Publisher, Edition, and AvailableCopies.

To create a Books table similar to the Readers table format in Microsoft Access:

Open Access and Create Table:

Go to the Create tab and select Table.

Switch to Design View.

Define Fields:

Add the following fields:

- InventoryNumber (Primary Key) – AutoNumber.
- Title – Short Text.
- CategoryID – Number (to link with a Categories table).
- Author – Short Text.
- Publisher – Short Text.
- Edition – Short Text or Number (depending on how editions are tracked).
- AvailableCopies – Number.

Save the Table:

Name the table Books and save it.

ReaderTypes Table: Fields should include ReaderTypeID and TypeLabel.

To create a ReaderTypes table in Microsoft Access, follow these steps:

Open Access and Create Table: Go to the Create tab and select Table.

Switch to Design View.

Define Fields: Add the following fields:

- ReaderTypeID (Primary Key) – AutoNumber.
- TypeLabel – Short Text (to describe the type, e.g., "Student," "Faculty," "Visitor").

Save the Table: Name the table ReaderTypes and save it.

BookCategories Table: Fields should include CategoryID and CategoryLabel.

To create a BookCategories table in Microsoft Access, follow these steps:

Open Access and Create Table: Go to the Create tab and select Table.

Switch to Design View.

Define Fields: Add the following fields:

- CategoryID (Primary Key) – AutoNumber.
- CategoryLabel – Short Text (to describe the category, e.g., "Fiction," "Science," "History").

Save the Table: Name the table BookCategories and save it.

Loans Table: Fields should include LoanID, ReaderID, InventoryNumber, LoanDate, and LatestReturnDate.

To create a Loans table in Microsoft Access, follow these steps:

Open Access and Create Table: Go to the Create tab and select Table.

Switch to Design View.

Define Fields: Add the following fields:

- LoanID (Primary Key) – AutoNumber.
- ReaderID – Number (to link with the Readers table).
- InventoryNumber – Number (to link with the Books table).

- LoanDate – Date/Time (to record the loan's start date).
- LatestReturnDate – Date/Time (to specify the due date for return).

Save the Table: Name the table Loans and save it.

2) Simple Input Validation Control

a) Define input masks for the "PhoneNumber" field:

In the Readers Table for the PhoneNumber field, set an input mask such as 000-000-0000 to ensure a standard phone number format with automatic dashes.

1. Open the Readers Table in Design View

Open your database in Microsoft Access.

Locate the Readers table.

Right-click on the table and select Design View from the context menu.

2. Select the PhoneNumber Field

In Design View, find the PhoneNumber field in the list of fields.

Click on the field to highlight it.

3. Set the Input Mask

With the PhoneNumber field selected, go to the Field Properties pane at the bottom of the screen.

Locate the Input Mask property and click in its cell.

4. Launch the Input Mask Wizard

Click the small ... button (ellipsis) next to the Input Mask property cell. This opens the Input Mask Wizard.

5. Choose a Phone Number Format

In the wizard, select Phone Number from the list of predefined masks. It will display the format !(999) 000-0000;0;- as an example.

Click Next.

6. Modify the Input Mask (Optional)

If you want a simpler format like 000-000-0000, type it directly into the input box and preview how it appears.

The wizard provides customization options, such as whether to include or omit placeholder characters like underscores.

7. Save the Mask

After confirming your desired format, click Next and then Finish to apply the input mask to the field.

8. Save the Table

Save the table by clicking the Save icon or pressing Ctrl + S.

9. Test the Input Mask

Switch to Datasheet View to test the PhoneNumber field.

Enter a phone number, and observe how the dashes (-) are automatically applied, enforcing the 000-000-0000 format.

Input Mask Behavior

Numbers Only: The mask restricts input to numbers in the 000-000-0000 format.

Validation: If the entered data does not match the mask, Access will prompt the user to correct it.

This ensures that all phone numbers in the Readers table follow a consistent and standardized format.

b) Validate non-negative values for certain fields:

For AvailableCopies in the Books Table, set a validation rule to ensure it’s non-negative: >= 0. Add a custom error message: "Number of copies cannot be negative."

To set a validation rule for the AvailableCopies field in the Books table to ensure it’s non-negative, follow these detailed steps:

1. Open the Books Table in Design View

Open your database in Microsoft Access.

Locate the Books table.

Right-click on the table and choose Design View from the context menu.

2. Select the AvailableCopies Field

In Design View, find the AvailableCopies field in the list of fields.

Click on the field to highlight it.

3. Set the Validation Rule

In the Field Properties pane at the bottom of the screen, locate the Validation Rule property.

Click in the Validation Rule field and enter the following rule: >= 0

4. Set the Validation Text

In the Field Properties pane, locate the Validation Text property (below the Validation Rule).

Click in the Validation Text field and enter the following custom error message: Number of copies cannot be negative.

5. Save the Table

Save the table by clicking the Save icon in the toolbar or pressing Ctrl + S.

If prompted, confirm any changes to the table structure.

6. Test the Validation Rule

Switch to Datasheet View to test the rule.

Try entering a negative number in the AvailableCopies field. Access will display your custom error message: Number of copies cannot be negative.

How It Works ?

The Validation Rule (>= 0) ensures that the value entered in the AvailableCopies field is zero or greater.

The Validation Text specifies a user-friendly error message displayed if the rule is violated.

This ensures that all entries for AvailableCopies are valid and align with the logical requirement that books cannot have a negative count.

c) Define non-null requirements for Name field:

Set Name in the Reader Table as required fields (Null not allowed) by setting the Required property to "Yes".

1. Open the Readers Table in Design View

Open your Microsoft Access database.

Locate the Readers table in the navigation pane.

Right-click on the table and select Design View from the context menu.

2. Select the Name Field

In Design View, find the Name field in the list of fields.

Click on the Name field to highlight it.

3. Set the Required Property

In the Field Properties pane at the bottom of the screen, locate the Required property.

Click in the Required property field, and select Yes from the dropdown menu or type Yes directly.

4. Save the Table

Save the changes to the table by clicking the Save icon or pressing Ctrl + S.

5. Test the Non-Null Requirement

Switch to Datasheet View to test the requirement.

Try leaving the Name field blank when adding a new record. Access will display an error message indicating that the field cannot be left empty.

How It Works?

The Required property ensures that a value must be entered in the Name field for every record.

If a user tries to leave the field blank, Access prevents the record from being saved until a value is entered.

By enforcing this rule, the database ensures data integrity by requiring that all readers have a name.

d) Create an editable list for the "City" field:

For the City field in Readers Table, create a Lookup Field with the options: "Oran," "Tlemcen,", "Mostaganem," and "Algiers".

1. Open the Readers Table in Design View

Open your database in Microsoft Access.

Locate the Readers table in the navigation pane.

Right-click on the table and select Design View from the context menu.

2. Select the City Field

In Design View, find the City field in the list of fields.

Click on the City field to highlight it.

3. Set the Field Type to Lookup Wizard

In the Data Type column for the City field, select Lookup Wizard from the dropdown menu.

The Lookup Wizard dialog box will appear.

4. Choose How to Get Values

In the wizard, select I will type in the values that I want.

Click Next to proceed.

5. Enter the List of Values

In the Column 1 box, type the following city names, one per row:

- Oran
- Tlemcen
- Mostaganem
- Algiers

Click Next when finished.

6. Save the Lookup Field

Select Finish to complete the Lookup Wizard setup.

The City field will now have a dropdown list in Datasheet View with the specified options.

7. Set the Field to Allow Edits

If you want users to add new cities beyond the predefined list:

In Design View, go to the Field Properties pane for the City field.

Set Limit to List to No.

If you want to restrict the choices strictly to the predefined list, set Limit to List to Yes.

8. Save the Table

Save the table by clicking the Save icon or pressing Ctrl + S.

9. Test the Dropdown List

Switch to Datasheet View.

Click on the City field for a record, and you'll see a dropdown list with the options: "Oran," "Tlemcen," "Mostaganem," and "Algiers."

If Limit to List is set to No, you can also manually type a city not in the list.

e. Default value for City:

Set the default value for City to Oran in the Readers Table.

1. Open the Readers Table in Design View

Open your database in Microsoft Access.

Locate the Readers table in the navigation pane.

Right-click on the table and select Design View from the context menu.

2. Select the City Field

In Design View, find the City field in the list of fields.

Click on the City field to highlight it.

3. Set the Default Value

In the Field Properties pane at the bottom of the screen, locate the Default Value property.

Click in the Default Value field and type the following: "Oran"

Ensure the value is enclosed in double quotation marks since it is a text value.

4. Save the Table

Save the changes to the table by clicking the Save icon in the toolbar or pressing Ctrl + S.

5. Test the Default Value

Switch to Datasheet View to test the default value.

Add a new record. The City field will automatically display Oran as the default value.

You can override this value by selecting a different city from the dropdown list or typing another name if the field allows it.

How It Works?

The Default Value property ensures that when a new record is created, the City field is automatically pre-filled with "Oran."

This is useful for minimizing data entry efforts if most readers are from Oran, while still allowing flexibility to change the value as needed.

3) Relationships between Tables

a. Define relationships with referential integrity:

1. Open the Relationships Window

- Open your database in Microsoft Access.
- Navigate to the Database Tools tab on the ribbon.
- Click on Relationships. A blank Relationships window will open, or you'll see an existing set of relationships if any have already been created.

2. Add All Tables to the Relationships Window

- In the Relationships window, click on Show Table (usually in the toolbar or right-click context menu).
- In the Show Table dialog box:
- Select the Readers, Loans, Books, ReaderTypes, and BookCategories tables.

- Click Add for each table.
- Close the Show Table dialog box after adding all the required tables.

3. Create Relationships Between Tables

For each of the specified relationships, follow these steps:

a. ReaderID in Readers Table → ReaderID in Loans Table

- Drag the ReaderID field from the Readers table and drop it onto the ReaderID field in the Loans table.
- The Edit Relationships dialog box will appear.
- Ensure both fields (ReaderID in Readers and Loans) are selected correctly.
- Check the box for Enforce Referential Integrity.
- Click Create.

b. InventoryNumber in Books Table → InventoryNumber in Loans Table

- Drag the InventoryNumber field from the Books table to the InventoryNumber field in the Loans table.
- In the Edit Relationships dialog box:
- Confirm the selected fields are correct.
- Check Enforce Referential Integrity.
- Click Create.

c. ReaderTypeID in ReaderTypes Table → ReaderTypeID in Readers Table

- Drag the ReaderTypeID field from the ReaderTypes table to the ReaderTypeID field in the Readers table.
- In the Edit Relationships dialog box:
- Confirm the selected fields.
- Check Enforce Referential Integrity.
- Click Create.

d. CategoryID in BookCategories Table → CategoryID in Books Table

- Drag the CategoryID field from the BookCategories table to the CategoryID field in the Books table.
- In the Edit Relationships dialog box:
- Verify the selected fields.
- Check Enforce Referential Integrity.
- Click Create.

4. Save and Verify Relationships

Once all relationships have been created, save the relationships diagram by clicking on the Save icon or pressing Ctrl + S.

Referential Integrity ensures that:

- You cannot add a record in a child table (e.g., Loans) if the corresponding record does not exist in the parent table (e.g., Readers or Books).
- You cannot delete a record in a parent table if it has related records in a child table unless Cascade Delete is enabled.
- This maintains consistent and valid data relationships between tables.

b. Fill each table with 4 sample records:

Manually enter data in Datasheet View or through data entry forms.

Readers Table

ReaderID	FirstName	LastName	ReaderTypeID
1	Amine	Dib	1
2	Anfel	Derbal	2
3	Ahlem	Larabi	1
4	Besma	Lallaoui	3

ReaderTypes Table

ReaderTypeID	ReaderTypeName
1	Student
2	Teacher
3	Student
4	Doctoral

Books Table

InventoryNumber	Title	Author	CategoryID
101	The Great Gatsby	F. Scott Fitzgerald	1
102	To Kill a Mockingbird	Harper Lee	2
103	1984	George Orwell	1
104	Moby Dick	Herman Melville	3

BookCategories Table

CategoryID	CategoryName
1	Fiction
2	Historical Fiction
3	Adventure
4	Science Fiction

Loans Table

LoanID	ReaderID	InventoryNumber	LoanDate	ReturnDate
1	1	101	01-12-2024	15-12-2024
2	2	102	02-12-2024	16-12-2024
3	3	103	03-12-2024	17-12-2024
4	4	104	04-12-2024	18-12-2024

4) Querying the database

a. List of all readers' phone numbers:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT FirstName, LastName, PhoneNumber FROM Readers;

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

b. List readers alphabetically by City, then by Name in descending order:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT FirstName, LastName, City FROM Readers

ORDER BY City ASC, Name DESC;

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

c. Information about readers in ORAN:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT * FROM Readers

WHERE City = 'Oran';

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

d. Query to delete readers from Tlemcen:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

DELETE FROM Readers

WHERE City = 'Tlemcen';

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

e. Titles of books in the "Computer Science" category:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT Title FROM Books

WHERE CategoryID = (SELECT CategoryID FROM BookCategories WHERE CategoryLabel = 'Computer Science');

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

f. Number of copies of books with the title "Access":

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT AvailableCopies FROM Books

WHERE Title = 'Access';

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

g. Total number of books:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT SUM(AvailableCopies) AS TotalBooks FROM Books;

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

h. Books with maximum and minimum copies, and their average:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT MAX(AvailableCopies) AS MaxCopies, MIN(AvailableCopies) AS MinCopies, (MAX(AvailableCopies) + MIN(AvailableCopies)) / 2 AS AvgCopies

FROM Books;

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

i. List of readers by city in descending order and name in ascending order:

Navigate to SQL View: Go to the Create tab and select Query Design.

In the Query Design window, click SQL View in the toolbar.

Write the SQL Command:

SELECT FirstName, LastName, City FROM Readers

ORDER BY City DESC, Name ASC;

Run the Query: Click Run (red exclamation mark icon) to execute the query and view the results.

5) Forms

a. Create a data entry form for books:

Create the Data Entry Form

Go to the Create Tab:

Click on the Create tab on the ribbon.

In the Forms group, click on Form Design. This will open a blank form.

Add the Books Table to the Form:

In the Form Design window, click on the Add Existing Fields button in the Design tab.

In the Field List pane that appears, find the Books table.

Drag and drop the following fields from the Books table into the form design area:

- InventoryNumber
- Title
- CategoryID
- Author
- Publisher
- Edition
- AvailableCopies

Set Up the CategoryID Field:

Since CategoryID is meant to link to another table (the Categories table), you can make it more user-friendly by using a Combo Box control.

Click the Combo Box button in the Design tab.

When prompted, select I want the combo box to look up the values in a table or query and choose the Categories table. This allows users to select a category from a list rather than typing in a number.

Arrange the Fields:

Once the fields are added, you can move them around to organize the form neatly.

Place the InventoryNumber field at the top (as it is an identifier), followed by Title, Author, Publisher, Edition, AvailableCopies, and finally CategoryID (the combo box for categories).

Customize the Form:

Optionally, add Labels for each field (e.g., "Book Title", "Author", etc.) to make the form easier to navigate.

Adjust the size and alignment of the text boxes to fit the layout.

Configure the Form for Data Entry

Set Form View: To make sure the form is ready for data entry, change to Form View by clicking on the View button in the Design tab and selecting Form View.

In Form View, the form will be ready to input new book data.

Save the Form

After setting up the form, save it by clicking File > Save or pressing Ctrl + S.

Name the form (e.g., "Book Data Entry Form").

Open the Form for Data Entry

To use the form, navigate to the Navigation Pane on the left side and double-click on the newly created form (Book Data Entry Form).

This will open the form, and you can start entering data for books.

Final Result:

Your Books Data Entry Form should allow users to:

Enter a Title for the book.

Select a CategoryID from a list of categories in the Categories table.

Input the Author, Publisher, and Edition.

Input the AvailableCopies.

The InventoryNumber will automatically be generated as an AutoNumber field.

The form will make data entry easier and more efficient, with users only needing to focus on filling in the fields and selecting from the combo box for categories.

b. Create an "Exit" button that allows you to close this form

Open the Form in Design View:

Add a Command Button: Go to the Design tab in the ribbon.

Select the Button control from the toolbox (it looks like a small rectangular button).

Click on the form where you want to place the button.

Use the Command Button Wizard:

When the Command Button Wizard appears:

Select Form Operations in the left panel.

Select Close Form in the right panel.

Click Next.

Set the Button Text or Image:

Choose how you want the button to appear: As text (e.g., "Exit") Or as an icon (choose an appropriate image if preferred).

Click Next.

Finish the Wizard:

Give the button a name (e.g., ExitButton) if prompted.

Click Finish to complete the setup.

Test the Exit Button:

Switch to Form View.

Click the button to ensure it closes the form.

6) Reports

First of all, we can create a SQL query that join the table Books to the table Category

SELECT * FROM Books, BookCategories

WHERE Books.CategoryID = BookCategories.CategoryID;

Name the request: BooksByCategory

Create a report listing books by category:

Go to Create > Report Wizard.

In the wizard, select the BooksByCategory query as the source.

Add fields to the report:

CategoryLabel, Title, Author, Publisher, Edition, AvailableCopies.

Group the report by CategoryLabel:

In the wizard, choose to group by CategoryLabel.

Sort the records within each group (optional):

For example, sort by Title in ascending order.

Choose a layout for the report:

Tabular layout works well for this type of report.

Finish the wizard and name the report, e.g., Books Report by Category.

Format the Report

Open the report in Layout View or Design View.

Adjust column widths and headings as needed.

Add a title to the report, such as "Books Report by Category".

Apply styles or themes to make the report visually appealing.

View and Export the Report

Switch to Report View or Print Preview to see the report.

Export the report:

Go to File > Save As > PDF or XPS to save the report as a PDF.

Alternatively, export to other formats as required.

7) Macros

To create a macro in Microsoft Access that opens a table and a form, follow these steps:

Step 1: Open the Macro Builder

Open your Access database.

Go to the Create tab on the ribbon.

Click Macro in the Macros & Code group. This opens the Macro Builder.

Step 2: Add an Action to Open a Table

In the Macro Builder, click the first row under the Action column.

Select OpenTable from the dropdown menu.

Configure the action:

Table Name: Choose the name of the table you want to open.

View: Select Datasheet (default).

Data Mode: Choose the mode, such as Read Only or Edit (default is Edit).

Step 3: Add an Action to Open a Form

Click the next row under the Action column to add another action.

Select OpenForm from the dropdown menu.

Configure the action:

Form Name: Choose the name of the form you want to open.

View: Select Form.

Filter Name: Leave blank unless you want to apply a filter.

Where Condition: Leave blank unless you want to specify a condition.

Data Mode: Choose Edit, Read Only, or Add depending on how you want to interact with the form (default is Edit).

Step 4: Save the Macro

Click the Save button (disk icon) in the toolbar.

Name the macro, e.g., OpenTableAndForm.

Close the Macro Builder.

Step 5: Test the Macro

In the Navigation Pane, find the macro you just created (under the Macros section).

Double-click the macro to run it.

Confirm that the table and form open as expected.

Optional: Attach the Macro to a Button

Open a form in Design View where you want to add a button.

Add a button to the form:

Go to the Design tab, select Button, and place it on the form.

When the Command Button Wizard opens:

Choose Miscellaneous > Run Macro.

Select your macro (OpenTableAndForm) from the list.

Complete the wizard, naming the button (e.g., "Open Table and Form").

Save the form and switch to Form View to test the button.

8) Interface

To create a general menu for the application, it will be necessary to follow the following steps.

Step 1: Plan the Menu

Decide what the menu will include, such as:

Opening forms.

Running queries.

Viewing reports.

Exiting the application.

Step 2: Create a New Form for the Menu

Go to the Create tab on the ribbon.

Click Form Design to create a blank form.

Save the form as MainMenu.

Step 3: Add Buttons to the Menu

Switch to Design View of the form.

Go to the Design tab in the ribbon and select the Button control.

Click on the form to place the first button.

Step 3.1: Configure the Button Actions

When the Command Button Wizard opens:

Select an action for the button. For example:

Form Operations > Open Form to open a specific form.

Report Operations > Open Report to open a report.

Application > Exit Application to close the database.

Select the object (form, report, etc.) you want the button to interact with.

Assign a label for the button (e.g., "Open Books Form").

Click Finish to create the button.

Repeat the steps to add more buttons for different actions.

Step 4: Customize the Menu Design

Add a Title: In Design View, use the Label tool to add a title like "Application Menu" at the top of the form.

Arrange Buttons:

Align buttons in a neat layout (use the alignment tools in the ribbon if needed).

Apply Styles:

Use the Property Sheet to apply background colors, fonts, or themes to make the menu visually appealing.

Step 5: Make the Menu the Default Startup Form

Go to File > Options.

In the Access Options window, select Current Database.

Under the Application Options section:

Set Display Form to Main Menu (or the name of your menu form).

Click OK and restart the database.

Step 6: Test the Menu

Open the database to confirm that the menu appears as the default startup form.

Click the buttons to verify that they perform the expected actions.

Conclusion

In today's digital era, databases play an essential role in storing, managing, and organizing large volumes of information. Database Management Systems (DBMS) provide the tools needed to efficiently handle data, ensuring its integrity, security, and availability. Understanding DBMS concepts is crucial for anyone working with data, as they form the foundation of modern applications and decision-making systems.

One of the key components of database management is SQL (Structured Query Language), which serves as the standard language for interacting with relational databases. SQL allows users to create, modify, query, and manage data efficiently. By mastering SQL, users can extract valuable insights, automate tasks, and optimize database performance.

Among the many database management tools available, Microsoft Access stands out as a user-friendly yet powerful system that combines a graphical interface with SQL capabilities. It enables users to design databases, create queries, generate reports, and develop applications without requiring advanced programming knowledge. Access is particularly useful for small to medium-scale database solutions, making it a great starting point for beginners while still being robust enough for professional use.

By understanding Databases, SQL, and Microsoft Access, users gain valuable skills applicable across multiple industries. These skills empower individuals to work with data-driven applications, business intelligence solutions, and automation processes, ultimately improving efficiency and decision-making.

As technology evolves, learning about databases remains an essential investment, opening doors to more advanced database systems like MySQL, PostgreSQL, Oracle, and SQL Server. Mastering these concepts will pave the way for deeper exploration into data science, big data, and cloud-based database solutions, shaping the future of information management.

In conclusion, whether for business, research, or personal projects, having a solid understanding of databases, SQL, and Microsoft Access equips individuals with essential problem-solving and data-handling skills, making them more effective in today's data-driven world.

References

[1] S. Alter, *Information Systems: Foundation of E-Business*, 4th edition. Upper Saddle River, NJ: Pearson College Div, 2002.

[2] J. Laudon et K. Laudon, *Management Information Systems: Managing the Digital Firm, Global Edition*, 17th edition. Harlow, England London New York Boston: Pearson Education Limited, 2021.

[3] C. J. Date, *An Introduction to Database Systems*, 8th edition. Boston, Mass.: Pearson, 2003.

[4] R. Elmasri et S. Navathe, *Fundamentals of Database Systems*, 7th edition. Boston Munich: Pearson, 2015.

[5] A. Oppel, *Databases DeMYSTiFieD*, 2nd edition. McGraw Hill, 2010.

[6] T. Connolly et C. Begg, *Database Systems: A Practical Approach to Design, Implementation, and Management*, 6th edition. Boston Columbus Indianapolis: Pearson, 2014.

[7] C. Coronel et S. Morris, *Database Systems: Design, Implementation, & Management*, 13th edition. Boston, MA: Cengage Learning, 2018.

[8] J. R. Groff, P. N. Weinberg, et A. J. Oppel, *SQL: The Complete Reference*, 3rd edition. New York, NY: McGraw Hill, 2009.

[9] « SQL Tutorial ». Avalable at: https://www.w3schools.com/sql/

[10] J. V. Petersen, *Absolute Beginner's Guide to Databases*, 1st edition. Que Pub, 2002.