People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University of 8 May 1945 -Guelma -
Faculty of Mathematics, Computer Science and Material Sciences

Department of Computer Science

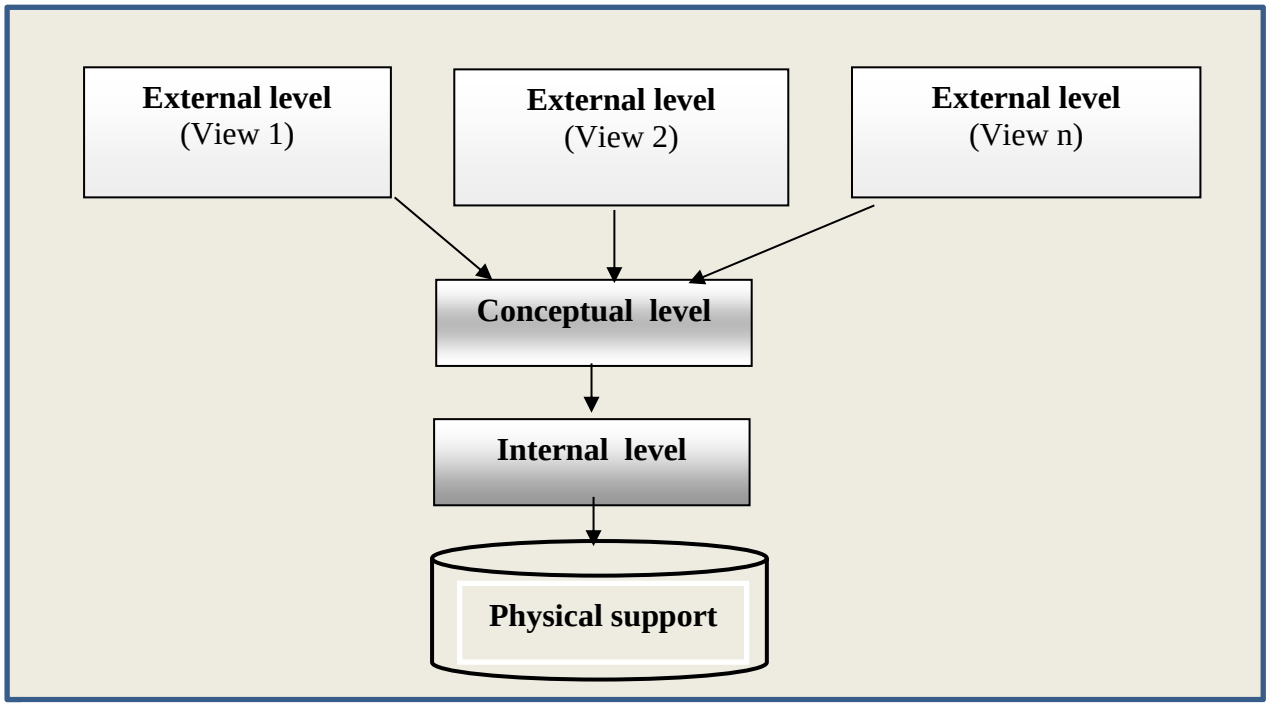
وزارة التعليم العالـي والبحث العلمـي
جامعة 8 مـاي 1945 قالمـة

كليـة الرياضيـات، الإعـلام الآلـي
وعلـوم المـادة

قسم : الإعلام الآلي

Course Handout

An Introduction to Databases



Intended for 2nd year computer engineering students
and
2nd year of undergraduate degree in computer science students

Academic Year: 2024-2025

by Dr. Ali KHEBIZI

Preamble

In the last few decades, the spectacular progress made in the field of **I**nformation and **C**ommunication Technologies (ICT) has led to a phenomenal explosion of the data mass generated daily by various stakeholders (companies, users, service providers, …etc.). This data is stored in appropriate supports of organizations' **I**nformation **S**ystems (IS), and it represents valuable information that supports decision-making. From a strategic point of view, the efficient exploitation of the wealth of information collected, and their efficient analysis are very useful to assist the companies' management systems in their decision-making activities, as well as the conduction of their daily business processes. However, several difficulties often arise and hinder the profitable exploitation of the collected data. These difficulties are due, in addition to their heterogeneous formats, to the fact that the data sources are diverse and are associated to different separate applications. In this perspective, the need of centralizing the handled data, structuring, and integrating them into a single reference structure, called a **Database**, are considered as inevitable activities to be conducted during the data management life-cycle. Indeed, this centralization aims to facilitate the future exploitation of data in the context of decision-making. Consequently, it is imperative to carry out preliminary actions in order to organize, model and ensure the consistency and integrity of the managed data.

In this course, we are particularly interested in the organization of data collected and stored in the information systems of organizations. The main objective is to allow the students to be aware of structuring and manipulating data in a tabular form. Thus, through the modeling of data in the form of a relational database, and by using the relational algebra and its underlying query language, they will be able to design relational models, verify their integrity and normalization, and then they can move to their exploitation, by deploying the SQL language.

Objectives of the course

The teaching of this course, entitled "**An Introduction to Databases (Courses & Exercises)**" aims to specify the needs and the interest of using databases, to show the issues related to their modeling and to expose the techniques aiming to structure them in a coherent and easy exploitable format. Finally, the interrogation of the conceived database by using relational algebra and the SQL language are deeply treated.
The course is intended for 2nd year computer engineering students and 2nd year of undergraduate degree in computer science students (LMD). It is consistent with the programs of the training courses proposed by the Ministry of Higher Education and Scientific Research of Algeria. Further, it can be used as a support whose use by various users (*directors, senior executives, support executives, developers, etc.*) can constitute a pragmatic and clear vision of the field of databases. Indeed, the content of the manuscript presents a concise and simple introduction to the principles of modeling relational databases and their related query language SQL. Hence, it allows the reader to learn the basic principles and techniques useful for the design, normalization, implementation and exploitation of databases in order to take advantage of the data handled by organizations. In this perspective, the course introduces a set of theoretical and practical knowledge to transform the mass of data, collected during the conduct of the company's business processes, into usable information. Thus, the information mine is materialized by a relational database whose exploitation, either interactively via the SQL language, or by the development of dedicated applications which use procedural languages, helps decision-makers to manage the company's resources and make informed decisions.

Studying this course allows the reader to acquire the following techniques:

• Understand the issues related to the diversity of data sources in companies and become aware

of the need to integrate them into a single homogeneous structure.
- Be able to analyze existing data sources, extract their content, then apply appropriate modeling techniques in order to handle their correlations and constraints while integrating them into a common database.
- Understand the techniques required for modeling and normalizing databases.
- Exploit the modeled database, both using algebraic language and SQL query language.
- Conduct a project to design a company's information system, focusing on the static aspect expressed through its database schema.

After studying this course, the student had to demonstrate the following skills:

i. Understand the principles of databases and be able to design a normalized relational schema.
ii. Create and manipulate a database using the Data Description Language (**DDL**)
iii. Be able to extract specific data from the database by manipulating, both relational algebra and SQL queries.

Structure of the manuscript

The manuscript is structured in an educational way that allows, firstly, the reader to get started with file management systems, while exposing their limitations in order to become aware of the problems related to data redundancy and the difficulty of their manipulation. Then, it explains the founding principles of databases with their levels of abstraction as well as database management systems. Secondly, the fundamental concepts and definitions useful for modeling relational databases are introduced and illustrated. To consolidate the designed relational schemas while minimizing data redundancies, normalization techniques and integrity constraints are presented and illustrated with different real case studies. In the perspective of an efficient exploitation of the conceived database, the relational algebra and its different operators are depicted. Finally, the practical implementation of each algebraic operator is translated into SQL language commands, and a detailed description of such SQL operators is highlighted in order to formulate queries satisfying various users requitements.

The manuscript is composed of four chapters, each of which addresses a particular aspect of the database field. At the end of each chapter, a series of exercises is proposed to students to allow them to consolidate and deepen the theoretical knowledge acquired in a specific aspect related to the concepts, models and languages proposed in the course. The proposed exercises are often real scenarios allowing addressing concrete problems related to the understanding, design and exploitation of databases.

We begin with **chapter 1** entitled "**Data bases presentation**" which highlights the limits of organizing data in the form of files and the imperative need to structure them in the form of databases. The different levels of abstraction and database management systems are presented and their functionalities are deeply described. Finally, the different data models are exposed and discussed.

In **chapter 2**, we tackle "**The Relational Model**", by presenting its various basic concepts, followed by the specification of the relation schema. This chapter deals in details with the theory of functional dependency and databases normalization, as well as the integrity constraints specification. It ends with the presentation of the notion of database schema.

The first two chapters introduce the material useful for understanding the principles of databases and their design. Then, we continue in **chapter 3** entitled "**The Relational Algebra**" in which we deeply discuss all the operators useful for manipulating databases organized in relational tables. We begin by presenting the unary operators (*selection*, *projection*), then we tackle the set operators.

In order to allow a practical application of the operators presented in the previous chapters, chapter 4, intitled "**The SQL Language**" is dedicated to the implementation and manipulation of the conceived database. To this perspective, the popular structured query language is introduced. The set of commands for creating, updating and querying the database are presented and illustrated. Furthermore, the algebraic operators are translated into the SQL language and the parameters of the commands are explained and discussed based on various use cases.

At the end of each chapter, a series of practical exercises is presented to the readers to allow them consolidating the most important concepts and to put into practice certain theoretical aspects acquired in the chapter. These exercises are either reviews of certain concepts and definitions whose mastery is essential for understanding the principles of databases, or real case studies that allow reinforcing the theoretical knowledge acquired in the current chapter. We end the handout with an appendix containing the detailed solutions to the series of exercises proposed in different chapters.

Finally, some bibliographical references are given at the end of the handout.

Distribution of the semester time volume over the chapters of the course.

Chapter	Chapter Title	Number of weeks
Chapter I	Databases Presentation	3 weeks
Chapter II	The Relational Model	4 weeks
Chapter III	The Relational Algebra	3 weeks
Chapter IV	The SQL Language	3 weeks

Table of Contents

Chapter I: Databases Presentation

1. Introduction

Since the early 1960s, organizations relied on file management systems (**FMS**) to handle the mass of manipulated data when conducting their daily tasks, such as payroll management, inventory management and accounting. These systems allow storing, retrieving, and manipulating data which was stored in flat files without a structured and consistent links between them. Although, file management systems were effective for their time, they began to show their limitations, due to the complexity of interconnected data and the increasing organizational needs. In fact, these systems, while groundbreaking in their early days, gradually revealed significant limitations as data volumes grew and applications became more complex, such as managing intricate financial records, tracking customer interactions, or supporting large-scale inventory systems. To overcome these limitations, databases systems have become the backbone of modern data management, revolutionizing how organizations handle information.

In this chapter, we start by introducing the notion of files, and then we will explore the drawbacks of file management systems. We will also highlight the transformative advantages of using databases. The chapter exposes the notions and definitions related to databases, as well as their associated database management systems (DBMS) with their functionalities. As data models constitute useful tools for representing data, the different existing models expressing data modeling are discussed and illustrated.

2. Notion of files

In the computer science discipline, data structures are fundamental concepts that enable the storage and manipulation of information. These structures range from arrays, linked lists, stacks and queues to trees, graphs and files.

Files structures provide a way to store data persistently on external storage devices underlying firms' information systems and allow applications to handle large datasets related to firms' different activities.

Hereafter, we give two definitions of the file notion.

Definition 1.1: *At the logical (or application) level, a file is a collection of related data (records) stored under the same name in a secondary storage memory, such as Hard Drive Disk (HDD), USB key or other external devices.*

Example

For instance, consider a file named "**`modules.txt`**". This file might contain:
Modules List:

```
- Algorithmic and data structures;
- Operating systems;
- Data bases;
- Information systems;
```

Here are some properties of the previous file:

- **File Name**: "**`modules.txt`**"
  - The **name** identifies the file.
  - The **extension** (.txt) indicates that it is a text file.
- **Content**: The data stored inside the file, which in this case is a simple list of modules.
- **Usage**: The file can be created, edited, or viewed using a text editor like Notepad.

Definition 1.2: *At the physical (system) level, a file is* structured sequence of bytes stored on a storage medium and *organized into blocks or sectors in secondary memory (SM). The data (records or commands) of the file are stored inside the blocks and are accessed using low-level mechanisms that bridge the gap between physical storage and the file management system.*

Example

Consider the electronic format of this course stored in its PDF format, named "**courseBDD.pdf**". The physical representation of this file on a HDD is as follows.

- **Data Blocks**: The file might consist of multiple blocks, each 4096 bytes (*typical block size*) constitute a separate block. For example, if the file is 1500 KB (1 536 000 bytes), it occupies 375 blocks.
- **Storage Location**: The operating system's file system maps these blocks to specific sectors on the HDD's platter. For instance:
  - Block 1 → Sector 1001
  - Block 2 → Sector 1002
  - Block 3 → Sector 1003
  - …….
- **Magnetic Encoding**: Each sector's data is stored as changes in magnetization on the platter's surface. A sequence like `10101010` might represent part of the file.

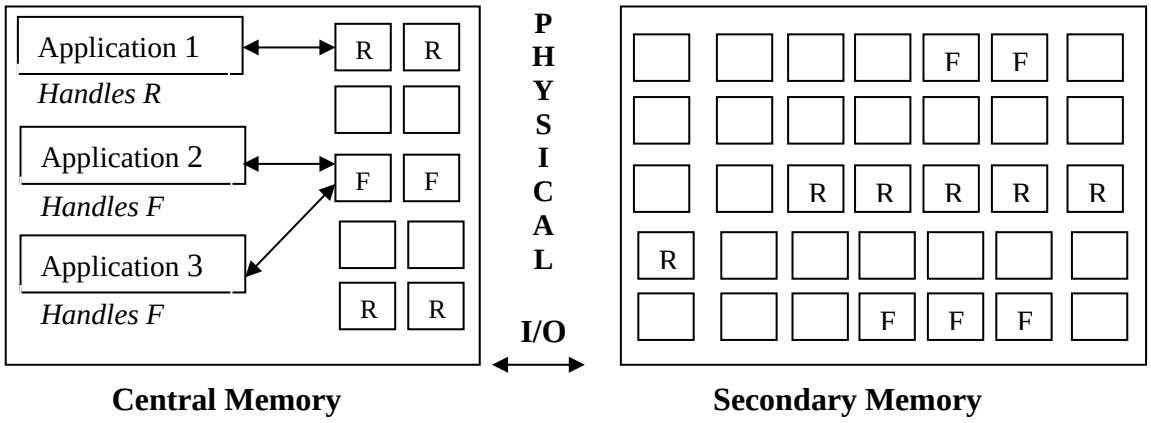


Figure 1.1 Data storage and manipulation using files structures

2.1 Interest of using files

By using files, the data mass is distributed among different files and used by different applications. Such a mechanism offers the following benefits.

- **Sharing and collaboration:** the used files can be shared among company members working on the same portions of data.
- **Flexibility**: files structures allow different data formats to be managed (*texts, images, videos or other formats*).
- **Integration with applications:** files can be used with a variety of software applications easily integrating with other systems and processes within the enterprise.
- **Security**: Files can be protected by security mechanisms (*passwords, access permissions and encryption systems*) to ensure the confidentiality of sensitive data and their integrity during transmission procedures.
- **Backup and Recovery**: to prevent data loss a copy of managed files is stored separately from the original source at regular intervals. The recovery process refers to restoring the backed-up files to their original or functional state following a data loss or corruption event.
- **Data analysis**: consists to manipulate the data stored in files in order to extract statistics and trends to support decision making activities.

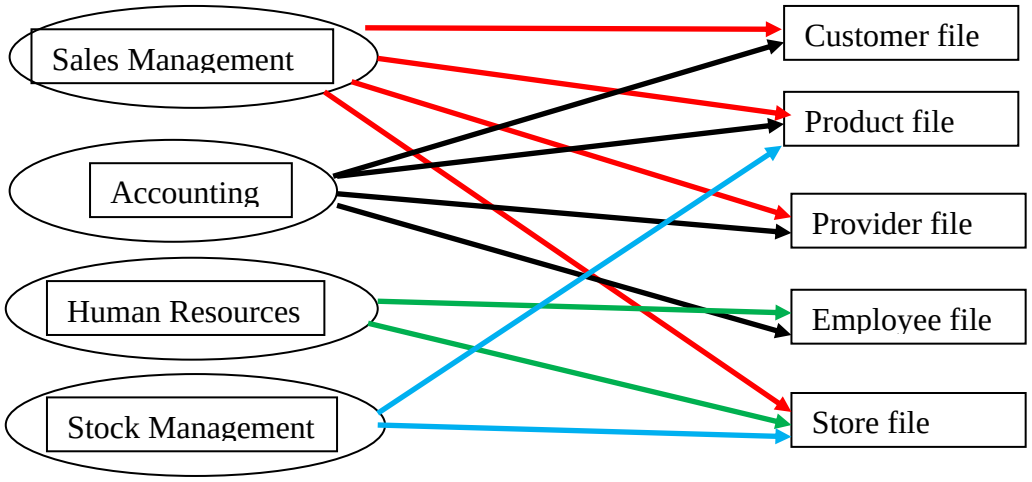


Figure 1.2 Using the data stored in files by software applications

The previous figure 1.2 highlights the interactions between different applications using diverses files.

For an efficient usage of files containing data, a specific module of the operating system, called *file management system* (**FMS**) is used**.**

Definition 1.3: *A file management system is a software module of the operating system which organize, store, retrieve and manage data stored in files.*

2.2 Limitation of files usage

Although they were initially used to exploit data by various management software, FMSs have several drawbacks. In fact, as it can be observed in the previous **figure 1.2**, some information is often stored in multiple locations. For example, the *product name* is stored in the *product file, the Provider file* as well as the *Store file*. This data duplication induces an increasing volume of data.

While they were widely used in early computing environments, FMSs have several limitations that are exposed bellow.

a. FMSs are limited when managing a large mass of data containing various links between them which leads to an increasing access time for an unplanned query.

b. Information redundancy issues: some data can occur more than once in different files. This fact leads to the following problems.

* Loss of memory space due to the multiple storage of the same information,
* Increased access time for searching and updating information due to the high data volume.
* Inconsistency of information: such a problem results if the identical information is replicated in multiple files multiple upadtes are required which leads to different conflicting values.

Example

Let's assume that the student address and its phone number describe at the same time; the students of the department, the resident of the university residence, the reader of the library and the player of sports activities. In this case, each change in the address or the phone number of a particular student must be operated in all the previous files, otherwise certain anomalies can be observed and the system can’t provide the correct address. Hence, a problem of information inconsistency can be observed if a multiplicity of updates is not operated.

c. The application is dependent on the data storage mode. In this case; the user is required to know:

- The access mode (*whether sequential, indexed or other*);
- The physical structure of the records and the location of the used files.

d. Data heterogeneity and weak integrity: due to the data storage in different formats. For example, the phone number has character type in the customer file and an integer type in the accounting file. Furthermore, difficulty of imposing constraints on data. For example, the product price must never be less than 0.
e. For each new user need a specific program is required.
   Example: If we want to estimate the evolution of sales during the last month according to the increasing rate 10 % of items' prices, a new programm is required.
f. Lack of security: in case of multiple access by several programmers to the same files, data security and unauthorized access are not guaranteed. Hence, it becomes very difficult to ensure data security and integrity. For example, certain employees should not have access to the payroll program.
g. Concurrency problem: if multiple users access the same files at the same time, then concurrency problems arise, leading to data inconsistency.
   Example: If T1 and T2 are two parallel transactions, the atomicity principle of transactions must be respected in order to avoid data inconsistency.
h. Integrating data from multiple files or sources is cumbersome and time-consuming, due to the variety of file formats and structures. Hence, the sharing process is complicated.

To overcome the previous limitations, **Databases (DB)** and their associated Databases Management Systems (**DBMS**) were developed in order to centralize and integrate data storage, eliminate redundancy, and provide powerful tools for data access, management, and security. The next section introduces and illustrates the notion of databases.

3. Introduction to the notion of database

By addressing the drawbacks of file systems, databases are designed to process large volumes of information, and they have become the backbone of modern data management, supporting applications in diverse fields, such as e-commerce, healthcare, accounting, and resources management, in general.

Definition 1.4: *A database is a persistent collection of structured, integrated and secured information.*

The term persistent means that the lifetime of data exceeds that of the execution of a program.

Definition 1.5: *A database is a collection of data which is structured independently of a particular application, consistent, with minimal redundancy, and accessible by multiple users.*

The **figure 1.3** bellow illustrates such aspects.

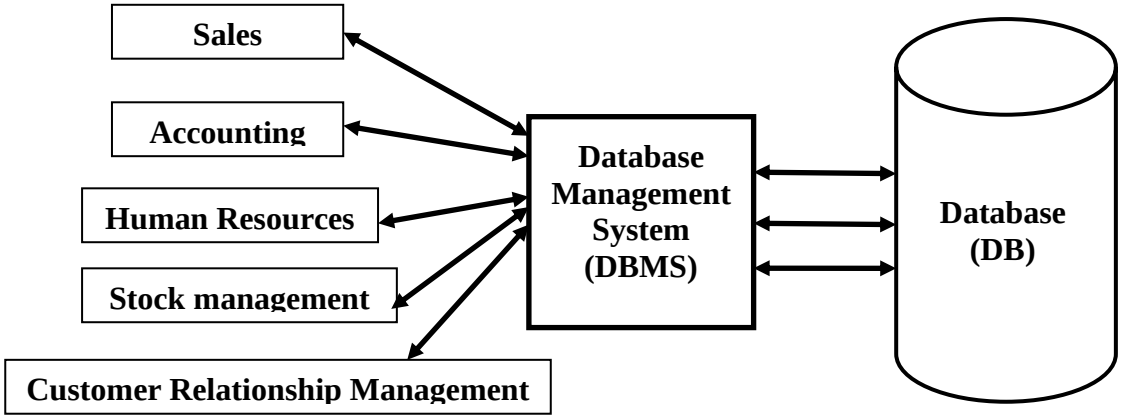


Figure 1.3 Accessing the database by the different applications

According to the previous definitions, a database is designed to handle large amounts of information efficiently and it supports various operations, such as querying, updating, and managing data integrity. Thus, it facilitates access, retrieval, update, and analysis of stored data.

The advantages of using databases are the following.

- Structuring data and facilitating the extraction and update of information.
- Manage data efficiently.
- Streamlining of access and processing procedures, as well as optimizing information processing.
- Ensuring the security of the information stored in the DB.
- Hide storage aspects by separating the logical level from the physical one (*Logical independence and physical independence*).
- Avoid conflicts resulting from shared data needing shared operations.

Examples of databases

- Company database: may contain data about customers, suppliers, products…
- Hospital database: manages information related to patients, doctors, rooms, services…
- Airline database: handles data about planes, flights, pilots…
- School database: Includes the data about modules, students, teachers, rooms, exams…

4. Abstraction levels of a database

To simplify the users' vision, and ensure abstraction of data stored on disks, the following three levels of data description have been defined.

- **The external level:** this level is relative to users and expresses how they perceive the data. Thus, it describes only the part of the data that is of interest to a particular user or group of users.
- **The conceptual level:** it's an intermediate level which corresponds to the "Abstract" representation of the entire DB and the managed data is considered semantically. Hence, this level allows describing all the company data, their links and their constraints.
- **The internal level:** reflects the physical memory and how the data is actually stored in the database. This technical level allows managing data storage on physical media, storage structures (files) and access modes (index management, keys, etc.).

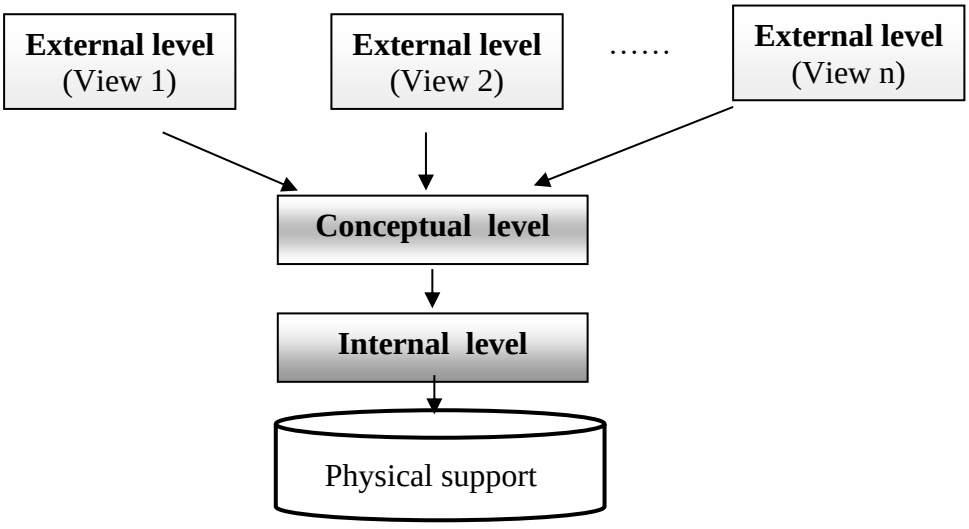


Figure 1.4 Illustration of the separate abstraction levels of a database

5. Databases Management Systems (DBMS)

As shown in **figure 1.3**, a software tool, named, **the database management system** (**DBMS**) is required to operate date stored in a DB. A DBMS acts as an intermediary between users and the database, allowing for efficient storage, retrieval, and manipulation of data while ensuring security and integrity.

Definition 1.6: *A Database Management System (**DBMS**) is a software used for creating, managing, and manipulating data. In this perspective it manages a set of files (DB), while ensuring users to store and extract data.*

More precisely, the DBMS performs the following functions.

- Create and update the DB structure by using the module Data Description Language (**DDL**).
- Manipulate and manage data by deploying the data Management Language (**DML)** module.
- Querying data of the DB for searching information of interest, by deploying the Data Querying Language (**DQL**) which allows viewing the target data.
- Administer the DB and operate the DB control by checking integrity constraints, logging, and transactions. These functions are ensured by the Data Control Language (**DCL**)
- DB security: access rights, confidentiality, recovery after failure.

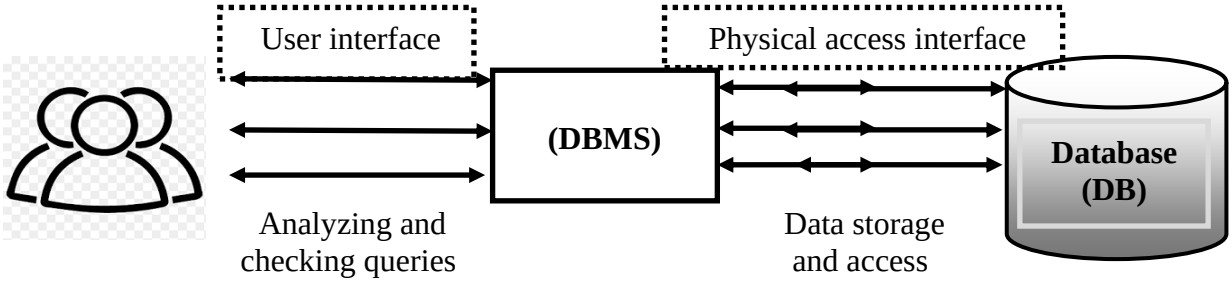


Figure 1.5 Interaction between users, the DBMS and the DB

Examples of industrial DBMS:

- DBMS on PC: Access, Foxpro, Paradox
- Freewares and Sharewares DBMS: MySQL, MSQL.
- DBMS on large Systems: Oracle, DB2, Sybase, SQL Server,

6. Types of data models

The design of a database schema requires a comprehensive and detailed description that is performed using a **data model** which expresses data constraints and their relationships. The concept of data model refers to a set of concepts that allows the database designer to build a comprehensive organizational representation of the company.

Definition 1.7: *A data model is a conceptual framework that allows designer to describe the data of a domain, while specifying their relationships, their constraints as well as their semantics.*

According to the previous definition a data model can be perceived as an intellectual tool for understanding the logical organization of data. More precisely, it consists of a set of concepts and rules for constructing a representation of reality using data types.

Using data models provides the following benefits to firms and database developers.

- **Efficiency:** Data models optimize data storage and retrieval processes.
- **Integrity:** Enforces rules to maintain data accuracy and reliability.
- **Standardization:** The data model ensures a consistent organization of data.
- **Communication:** It provides a clear blueprint for developers, analysts, and stakeholders.

Several types of data models have been used and many of them continue to be deployed to build databases. In what follows, we will explore these different models. For each one, we give its definition and key features, we illustrate it with a real-word example, and then we evaluate its strengths and weaknesses.

6.1 Hierarchical model

In this class of models, data is organized into a tree-like structure with parent-child relationships. A virtual root that serves as an initial structure holds pointers to sub-trees. Each child node has a single parent, and relationships are represented as links between nodes. Such models were intensively used in early database systems like IBM's Information Management System (IMS).
Example: Consider a company that needs to store its organizational hierarchy, including departments, teams, and employees.

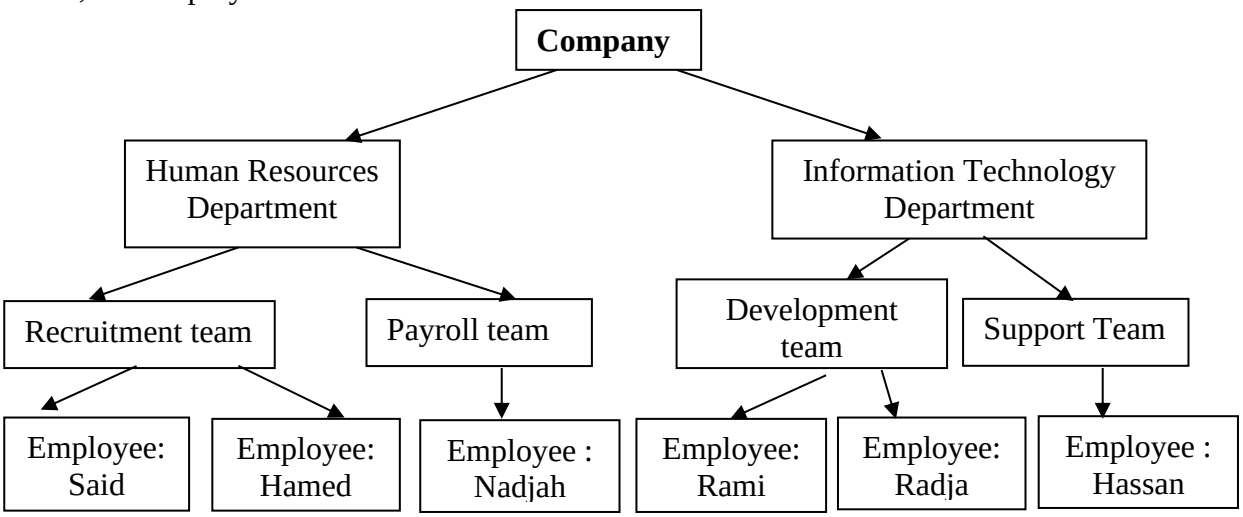


Figure 1.6 Illustration of a hierarchical data model of a company

In the previous figure the root node corresponds to the company, the parent ones are the departments within the company (*Human resources and information technology*), while the child nodes express the teams under each department and the leaf nodes represents employees under each team.

Advantages

- It's an intuitive model useful for real-world hierarchical structures.
- The model is ideal for queries that require navigating parent-child relationships due to its fast traversal (*e.g., finding all employees in a department*).
- It ensures data integrity by strictly enforcing compliance with parent-child relationships.

Disadvantages

- A child can only have one parent, which may not reflect some real-world scenarios (*e.g., an employee working in multiple departments*).
- Data are replicated.
- Modification of information on all occurrences during an update;
- All queries depend on the root. Hence, queries that require traversing across branches or relationships outside the hierarchy are cumbersome, because users have to traverse the tree starting from the root to find the information.

6.2 Network model

The **network data model** extends the hierarchical model by allowing many-to-many relationships between nodes. It allows for cross-sectional relationships (*i.e., many-to-many relationships between related records*). Unlike the hierarchical model, in which each child node has only one parent, the network model supports complex interconnections, enabling greater flexibility. Thus, a record can be a member or child in multiple sets. Consequently, this model allows the use of complex structures, and it contains entities that have the characteristics of a record. Further it manipulates pointers that ensure the links between the different records.

Example: A university wants to manage its data about departments, professors, courses and students which have complex relationships between them.

- Professors can teach multiple courses (*i.e., Professor A teaches course1 and course 2*).
- A course can be taught by multiple professors (*i.e., Course 2 is ensured by Professor A and Professor B*).
- Students can enroll in multiple courses (*i.e., Student 10 enrolls course 1, course2, course 5 and course 6*).
- A course can have many students (*i.e.,course 6 is enrolled by students 10, 30 and 50*)..

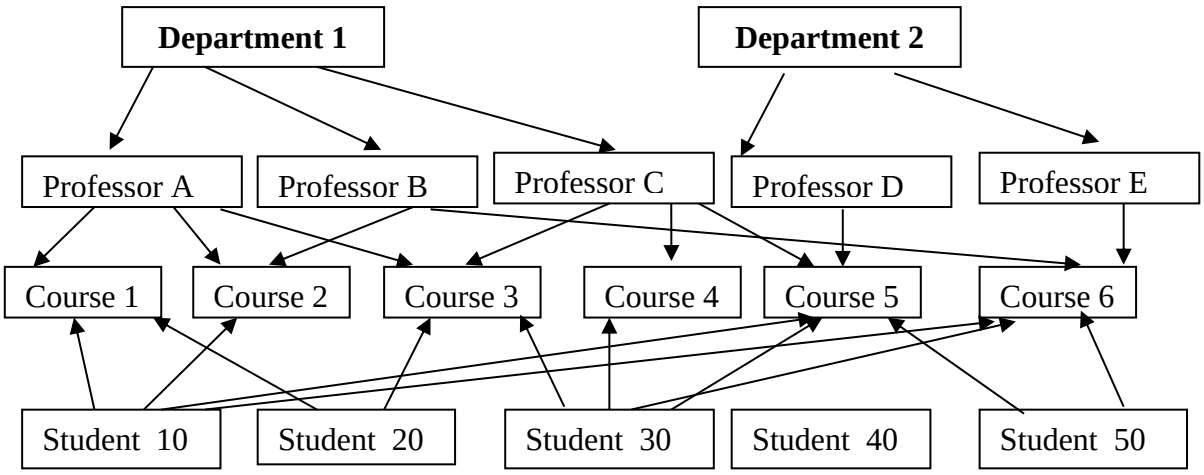


Figure 1.7 Example of a network data model for handling the university data

As it can be observed in the previous figure, the network data model is ideal for scenarios with complex, interconnected data by enabling richer relationships and efficient queries across multiple levels of interconnected entities.

Advantages

- The model supports many-to-many relationships natively and diversifies access to a group of data.
- Avoids data repetitions by only replicating pointers.
- Data organization is optimized for traversing relationships, such as querying courses taught by a professor and attended by students.
- The model is more flexible and well adapted to scenarios with interconnected data.

Disadvantages

- Complex structure to define and more challenging to design compared to the hierarchical model.
- It requires a sophisticated database management system (*e.g., CODASYL DBTG model*).
- Can cause serious problems if a data has been forgotten when defining the problem.

6.3 Semantic model

Semantic data models capture the meaning and relationships between data elements, providing a more complete and meaningful representation of the data. This enriched context enables more accurate interpretation, querying, and analysis of the data. It emphasizes the relationships between data and their real-world meanings, making it especially suitable for complex and interconnected systems. In this perspective, it models real-world objects as entities with attributes that describe their properties, and focuses on the meaning of the data rather than just its structure, ensuring that the model aligns closely with the domain it represents.

A semantic model is represented with a directed graph. Its vertices represent the concepts, and the links between the vertices (*nodes*) represent the semantic relations, connecting the lexical fields.

Example: Below is a complete example of a semantic data model expressing courses management.

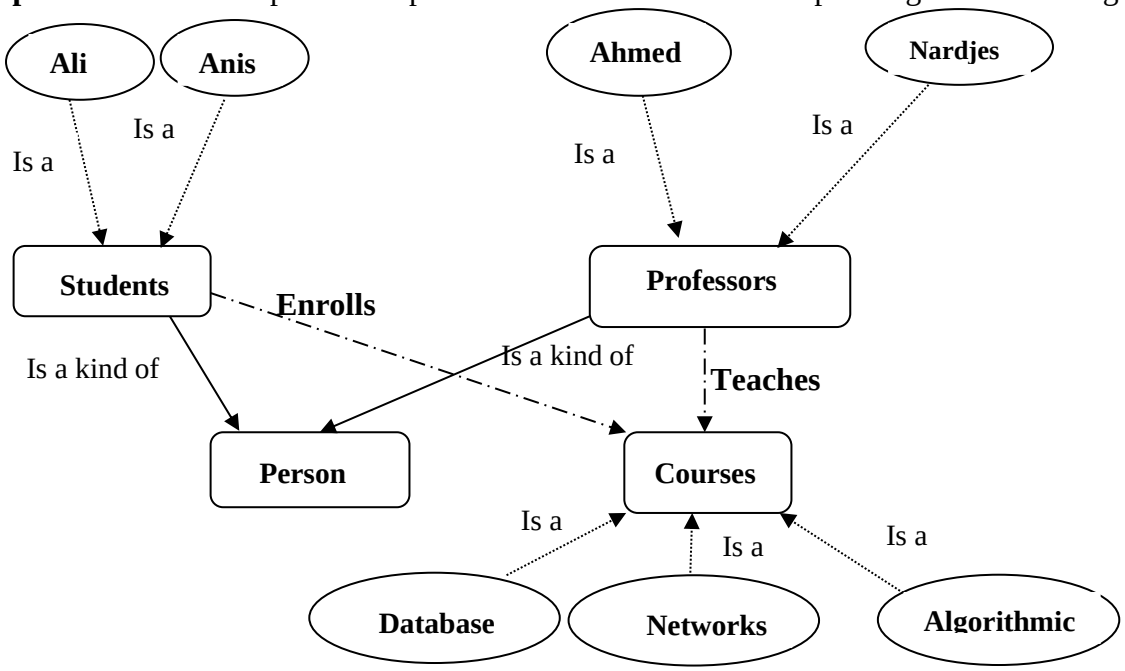


Figure 1.8 Illustration of the semantic model

Legend

Generic concepts — Individual concepts

Link "is kind of " — Link "is a" — Semantic links

Advantages

- As a high-level data modeling technique, this model captures the semantics (*meaning*) of data by modeling real-world entities and their relationships in a more natural and intuitive way.
- By emphasizing the relationships between data and their real-world meanings, the model is especially suitable for complex and interconnected systems.
- Aligns closely with the vocabulary and logic of the domain, making the model easier to understand for domain experts.
- It offers a global accessibility by providing standardized ways to represent and share knowledge, making it accessible and reusable across systems.

Disadvantages

While the semantic data model provides a richer and more expressive framework compared to the two previous ones, it also has several disadvantages that can limit its practicality in certain scenarios. The main limits of the model are exposed bellow.

- Designing a semantic data model requires a deep understanding of the domain, including intricate relationships, rules, and constraints. Thus, it takes significant time and expertise to create a comprehensive semantic model, which may not be feasible for smaller projects.
- Complex relationships and hierarchies may result in slower query performance compared to simpler models like relational databases.
- Lack of uniformity and the absence of universally accepted standards for semantic data modeling, which can lead to inconsistencies in implementation and understanding.
- The implementation and management of the conceived models are more challenging, due to the fewer tools available compared to relational databases.

6.4 Object model

The object data model performs a junction between object-oriented programming concepts and database management area. Hence, data is stored as objects with their attributes and methods. As an immediate consequence, object-oriented programming principles, such as encapsulation, inheritance, and polymorphism can be applied in this context. Thus, the object model organizes data as objects that encapsulate attributes (*data*) and methods (*behavior*). This model is particularly useful for applications that deal with complex and interrelated data.

Example 1: In the CRM (*Customer Relationship Management*) field, we can identify the following entities: **customers, products, orders,** and **payments**. Using the object-oriented model, each entity is represented as a class, and instances of these classes are perceived as objects.

The relationships linking the previous classes are the following:
- A **Customer** can place multiple **Orders**.
- An **Order** can contain multiple **Products**.
- An **Order** is associated with a **Payment**.

The **Table 1.1** bellow depicts the different classes with their attributes and methods.

Classes	Attributes	Methods
Customer	- CustomerID (*Unique Identifier*) - Name - Email - Address	- PlaceOrder() - ViewOrderHistory()
Product	- ProductID (*Unique*) - Name - Description - Price - StockQuantity	- UpdateStock() - ApplyDiscount()
Orders	- OrderID (*Unique*) - OrderDate - Customer (*Object Reference*) - Products (*List of product objects*) - TotalAmount	- CalculateTotal() - AddProduct()
Payment	- PaymentID (*Unique Identifier*) - Order (*Object Reference*) - PaymentMethod - PaymentStatus	- ProcessPayment()

Table 1.1 Description of the object data model for handling CRM activities

Example 2: The object model expressing the store management of a company is described below.

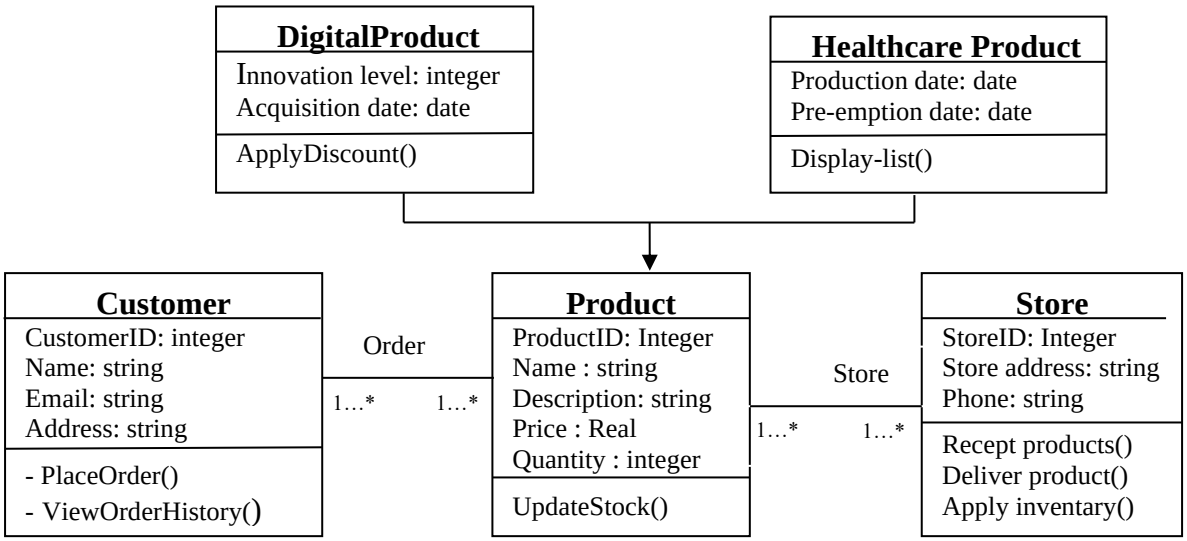


Figure 1.9 An example of object data model for managing stores

Advantages

Using object-oriented data-models offers the following benefits for data managers.

- The model **s**upports hierarchy and shared behavior (e.g., a *DigitalProduct* class inheriting from *Product*).
- The model handles relationships and nested data efficiently. Thus, it performs complex data representation.
- It's very ideal for real-world scenarios due to its abilities to combine data and behavior in the same schema.

- Classes and methods can be reused across the applications. Consequently, the development cycle of software and databases design are optimized.

Disadvantages

While the object-oriented data model is powerful and flexible for certain domains, like multimedia systems, and real-time systems, its disadvantages make it less suitable for simpler or large-scale transactional applications. Bellow, the weakest points of the model are listed.

- Object-oriented models can become overly complex due to the use of advanced concepts like inheritance, polymorphism, and object references.
- Inheritance and encapsulation can sometimes lead to data redundancy.
- Designing the schema requires a deep understanding of both object-oriented principles and database design, which may increase the learning curve.
- Variations in object-oriented database systems make interoperability and migration between platforms challenging. In fact, the standardization efforts are limited.
- Performing complex queries that span multiple objects and relationships can be cumbersome and require custom query languages or APIs. Further, handling large datasets with deeply nested objects can lead to performance degradation.

6.5 Entity-Association model (E/A)

The Entity-Association data model is a high-level conceptual model which is used for designing relational databases. Hence it focuses on data organization and relationships rather than implementation details. It's based on three concepts: entities, attributes, and relationships.
The entities express the elements or objects managed in the studied area, such as products, stores and customers for the CRM domain. These entities are described by a set of properties or attributes, and are linked together with relationships expressing the semantic relation between entities.

Example: Let us consider the field of schooling and the activity of student assessment in the different types of modules. The modeling of this area is depicted in the following data model.

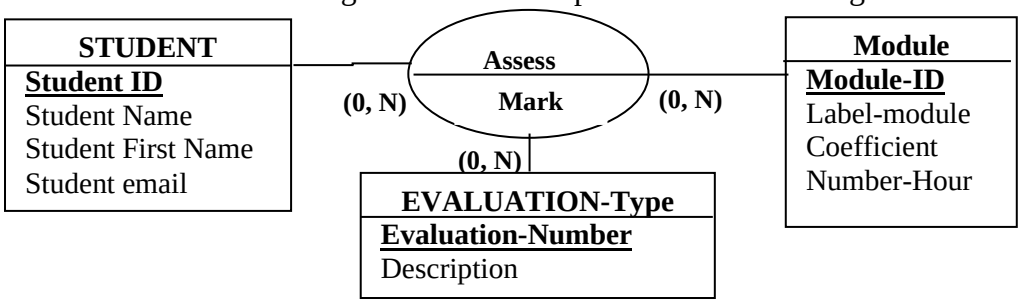


Figure 1.10 Example of an Entity-relationship data model

Advantages

- The E/A data model provides a conceptual and abstract view of the database which helps developers visualize and understand easily the structure of the database.
- The clear representation offered by the E/A model facilitates communication between technical and non-technical team members due to their easy to interpret.
- It's a versatile model that can model complex relationships, including one-to-one, one-to-many, and many-to-many. Further, it supports additional constructs such as weak entities, aggregation, and specialization/generalization.

- The E/A model is very adaptable because it can be easily extended or modified as system requirements change. In such scenarios, new entities, attributes, or relationships can be added without disrupting the entire design.

Disadvantages

- The model lacks of implementation details, and it does not address physical storage or performance optimizations.
- Requires translation into a physical data model (*e.g., relational schema*) for implementation.
- It's a time-consuming model, as creating a detailed E/A model requires careful analysis and can be time-intensive, especially for large systems with complex requirements.
- As the E/A model was primarily designed for structured data with clear entities and relationships, it's not suitable for unstructured data.

6.6 Relational model

To overcome the shortcomings of the previous data models, especially, the network and hierarchical models, while ensuring data integrity and avoiding data redundancies, the relational model has been elaborated. This model organizes data into **tables** (*also called relations*), which consist of rows (*records*) and columns (*attributes*). It provides a rigorous and flexible foundation for data representation and manipulation.

Example: Let's consider the inventory management database.

The database contains three tables: ***Product***, ***stores*** and ***storage*** described with the following attributes.

- **Product**: Product ID, name, unit price
- **Stores**: Store-ID, description, phone, address
- **Storage**: Store-number, Product-ID, quantity

The following tables depict an instantiation of the different table of the database.

Product table

Product-ID	Name	Unit-price
100	Pen	50
200	Ruler	60
400	Markers	100

Store table

Store- ID	Description	Phone	Address
A	Central stores	0665847841	Guelma
B	Annex store	0578487412	Annaba
C	Market store	0784789621	Guelma

Storage table

Product ID	Store- ID	Quantity
100	A	2000
400	A	5000
400	C	150

Figure 1.11 Tables and occurrences of the relational inventory database

Advantages

- The relational model has a sound formal foundation, because it's based on the mathematical principles of set theory and the predicate logic one.
- The tabular structure is intuitive and easy to understand, thus leading to a simple and intuitive model.
- The model is flexible due to the usage of relationships between tables that makes it suitable for a variety of applications.
- Enforced through primary and foreign keys, ensuring consistent, and data integrity.
- By supporting large datasets with indexing and optimization techniques, the model is very scalable.
- Manipulation of the BDD by high-level non-procedural languages (*SQL: Structured Query Language*).

Disadvantages

- Relational models often normalize data, breaking it into multiple tables in order to maintain data integrity. While this avoids redundancy, it increases storage requirements and query complexity.
- Queries involving multiple joins across large tables can degrade performance significantly, especially for complex relationships.
- Altering schemas in production can be time-consuming and risky. In fact, the model uses a predefined schema, making it difficult to adapt to evolving requirements without significant restructuring. For example, when adding a new attribute or changing an attribute type.
- Relational data model is not suitable for managing **unstructured** or **semi-structured data** such as images, videos, documents, or JSON objects.

7. Conclusion

With the spectacular development of computing and its democratization, the volume of daily generated data increased considerably, inducing increased complexity in the manipulation of the produced data. Thus, the need for more robust solutions, which exceeded the limitations of FMS, led to the advent of database management systems (DBMS) in the 1960s.

Although the first generation of data models (*hierarchical and network*) offered a well-structured organization of data, these models lacked flexibility and required complex navigation queries.

A paradigm shift towards the relational database model emphasized data independence, where the logical organization of data was decoupled from physical storage mechanisms. Relational databases structured data into tables with well-defined relationships, allowing for greater flexibility, scalability, and ease of use.

Basing on its simplicity and its solid theoretical foundation, the relational model addressed the major shortcomings of previous systems, minimizing redundancies and improving data integrity. Over the decades, relational databases have become the standard for transactional applications and continue to dominate data storage.

The next chapter will be dedicated to the study of the relational database model and its foundation.

Series of exercises No. 1

Exercise 1 : The file management systems (FMS) present different limitations.

a. Recall the main drawbacks of FMS?
b. How can such limitations be overcome ?
c. What are the consequences of data redundancy ? Illustrate your pupose with a real-world example.

Exercise 2 : Database fundation is based on the principle of separating the data manager vision into different abstract levels.

a. What is the purpose of separating the view of databases into several levels of abstraction ?
b. What is the conceptual level and how is it approached ?
c. Illustrate the external level with a real word scenario ?

Exercise 3 : Draw a comparison between network and hierarchical data models, basing on the following features: structure, flexibility, relationship, query execution.

Exercise 4 : DBMS are required to ensure the interaction betwwen data users and the database.

a. Give a brief description of the different languages performed by a DBMS?
b. Suppose that a particular DBMS crashes and becomes unable to ensure database security. What are the consequences of such a crash?
c. Suggest different solutions to avoid such situations?

Exercise 5 : Although the E/A data model provides a sound framework for the database modeling, it is not implementable in a commercial DBMS.

a. What are the concepts manipulated by this model?
b. What types of relations are handled by the E/A data model?
c. How the E/A data model is exploited to manage data when using a DBMS?

Chapter II: The Relational Model

1. Introduction

As explained in the previous chapter, database's systems have emerged as a solution to the problems raised by FMSs. To this perspective, several types of data models have been proposed to allow efficient representation of the set of data to be stored in databases. Each model offers certain advantages and suffers, at the same time, from certain limitations. In particular, the relational model, based on a very solid theoretical foundation, constitutes the fruit of the historical evolution of data models.

This chapter is devoted to the presentation of the relational model. Its basic concepts are defined and illustrated, and its underlying theory is highlighted. Thus, we will deeply discuss the functional dependency principles and the normalization theory, as well as the integrity constraints. A special attention is paid to the transformation of the E/A model to the relational one. We end the chapter with a conclusion.

2. Definition of the relational model

The relational model was initially proposed by E.F. Codd in the early 1970s with the primary goal of increasing program independence from data representation. It is based on a very strong theoretical foundation combining the set theory, first order predicate theory and functional dependency with the normalization theory.

Hereafter, we expose an explicit definition of the relational model.

Definition 2.1: *A relational database is a set of relationships that is variable over time. Each relationship is seen as a table whose columns designate the attributes (fields, properties) and the rows designate the tuples or records.*

Illustrative example: A company is organized into departments which are themselves composed of branches. A branch belongs to one and only one department and each department is headed by a single manager among the employees of the company. An employee is assigned to one and only one department, but there are employees without a department. The company carries out projects and in which participate several employees.

The E/A model of the previous enterprise is depicted in the **Figure 2.1** bellow.

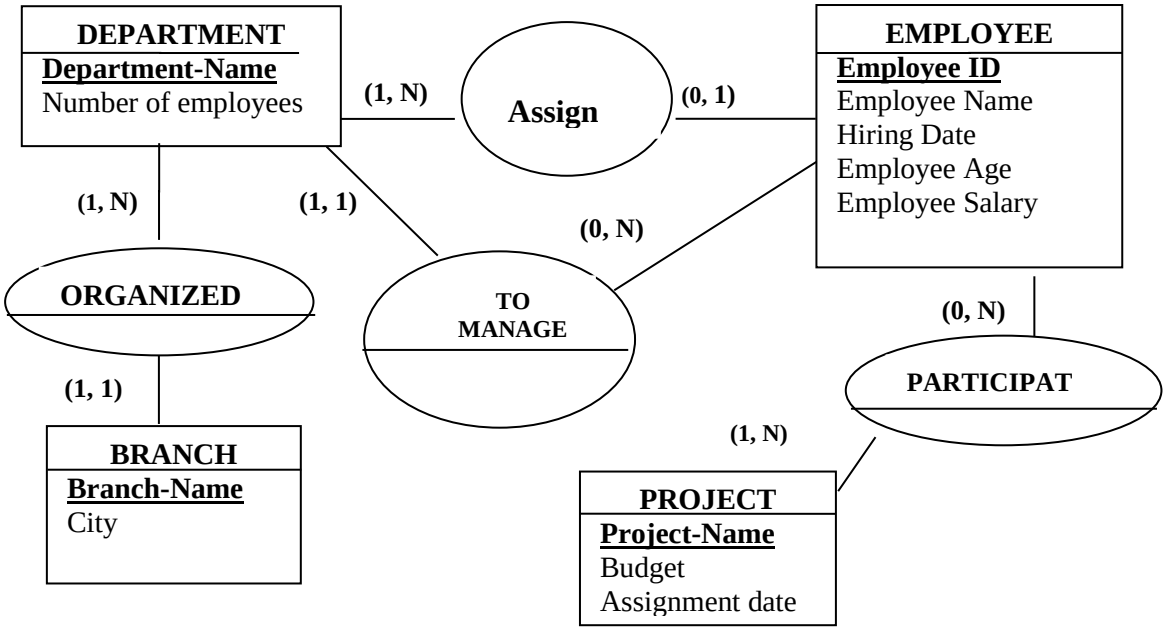


Figure 2.1 The Entity-Relation data-model of the company

The previous model is an abstract representation of the company. As shown in **Figure 2.2** bellow, the corresponding relational database of the company is described by the following set of tables, each of which contains various rows and columns.

Table **Employee**

Employee-ID	Employee name	Hiring date	Employee age	Employee salary	Department name
100	Mohamed anis	01/01/2010	45	45.000,00	Construction
200	Kamel Salem	15/07/2015	30	70.000,00	Construction
300	Ramzi saad	07/03/2022	41	83.000,00	Accounting

Table **Department**

Department name	Number of employees	Employee-ID
Construction	200	100
Research and development	5	200
Accounting	8	200

Table **Branch**

Branch name	City	Department name
Building	Guelma	Construction
Bridges	Annaba	Construction
IT	Annaba	Research and development

Table **Project**

Project name	Budget	Assignment date
Road A	2.000.000,00	17/01/2023
Building C	50.000,00	14/10/2024
Innovation IT	800.000,00	07/08/2020

Table **Participate**

Project name	Employee-ID
Road A	100
Innovation IT	200
Road C	100
Innovation IT	300

Figure 2.2 Relational tables expressing the company's data

3. Basic concepts of the relational model

Like any model, the relational one is based on a set of concepts that allow constructing a coherent representation of the database. These concepts are described and illustrated below.

3.1 Domain

A domain corresponds to a set of values taken by an attribute. This set of values can be finite or infinite, and it can be specified in two different ways.

a. By extension: in this case, all the possible values of the attribute are listed.

Example

Basic colors: {Red, Blue, Yellow},
Sex: {Red, Blue, Yellow},
Days of the week: {Mon, Tue, Wed, Thur, Fri, Sat and Sun}.

b. In intention: in this case, only the formula or the rule describing the attribute is given.

Example

Mark: [0,20],
Price: [100, 5000],
Name: String,

3.2 Attribute

The attribute refers to the elementary data manipulated in the field of study. It specifies the role of the considered property in the considered relation. The attribute takes its value in a domain and corresponds to the columns of the relation.

Example

The attributes of the relation ***EMPLOYEE*** of the **figure 2.2** are the following.

- *Employee-ID*: describes the employee unique identifier and takes its values in the interval [001,1000].
- *Employee name*: consists of a string of characters designating the employee's name.
- *Hiring date*: is a particular date reflecting the hearing data of the employee, and it's included in the interval [01/01/2000…31/12/2025].
- *Employee age:* Corresponds to an integer expressing the employee age and it takes its values in the domain [18,60].
- *Employee Salary:* this attribute refers to the monthly salary amount and it's included in the interval [10.000, …, 200.000].
- *Department name:* this attribute is a foreign key specified as a string taking its values in the set the finite set of departments: {Construction, Research and development, Accounting}

3.3 Tuple

A tuple (*row or record*) in a relation expresses a set of values taken by the attributes of the considered relation. Thus, each tuple records the separate values of the columns defined in the table.

Example

The relation *Branch* of the **figure 2.2** contains the three following tuples.

- Building, Guelma, Construction.
- Bridges, Annaba, Construction.
- IT, Annaba, Research and development.

3.4 Relation

A relation is a subset of the Cartesian product of a list of domains. More formally.

An n-ary relation on the domains D_1, D_2, …, D_n is a subset of the Cartesian product $D_1 \times D_2 \times \ldots \times D_n$. Consequently, the relation is composed of a set of tuples determined by a name, and which can be expressed in intension (*schema*) or in extension (*tuples or occurrences*).

Example

Let us consider the four domains of the school management area related to students. In this area, the following four domains are defined.

Registration number, First name, Second name and Average.

D_1 (registration number): Integer [1…50000],

D_2 (First name): string,

D_3 (Second name): string,

D_4 (Average): [0…20],

Now, we can specify the relation ***Student (Registration number, First-Name, Second-Name, Average)*** which expresses the subset of the following Cartesian product:

Registration number x ***First-Name*** x ***Second-Name*** x ***Average***

The previous relation ***Student (Registration number, First-name, Second-Name, Average)*** defines the *schema* or the *intension* of the relation. This extension of the relation is a bi-dimensional table composed of a set of tuples, each of which is composed of the four attributes.

Student ← Relation Name

Registration number	**First-name**	**Second-Name**	**Average**
101	Mahmoudi	Salah	15,15
3154	Djeribi	Wassila	10,63
413413	Soltani	Malik	9,40

Attributes of the relation

Tuples of the relation

Table 2.1 Illustration of the basic concepts: attributes, tuples and relation name

3.5 Schema of a relation

The schema of a relation represents its intension and is expressed by the name of the relation followed by the list of attributes and their application domains. Hence, it specifies the common and invariant properties of the tuples that the relation will contain. As, the schema is an abstract representation, thus the relation's list of tuples is not listed.

Example: Hereafter two examples of relation's schema are shown.

CUSTOMER (Client-ID: *integer*, First-Name: *String*, Last-Name: *String*, Birth-Date: *Date*).
PRODUCT (Product-code: *integer*, Product-label: *String*, Unit-Price: *real*).

For clarity and simplicity writing reasons, the relation schema is limited to the name of the relation followed by its list of attributes and the domain names are omitted.

The previous two relations schema become:

CUSTOMER (Client-ID, First-name, Last-name, Birth-date).
PRODUCT (Product-code, Product-label, Unit-price).

4. Transition from the Entity-Association (E/A) model to the relational model

The E/A model exposed in chapter 1 (***See § 6.5***) is a powerful conceptual tool which is very useful for elaborating a data model expressing links between concepts manipulated within a particular domain. However, as it's an abstract representation of the reality it can't be directly implemented in SGBD environments. Consequently, E/A data models must be converted to relational models in order to be handled by relational SGBD. To this end, a set of transition rules allowing the transition for E/A models to relational model are defined. The set of transition rules are given bellow.

Rule 1: Any entity in the E/A model becomes a relation in the relational model. In the obtained relation the entity identifier becomes the primary key of the relation and the attributes of the entity become attributes of the relationship.

Example

PATIENT
SSN
First-name
Last-name
Birth-date

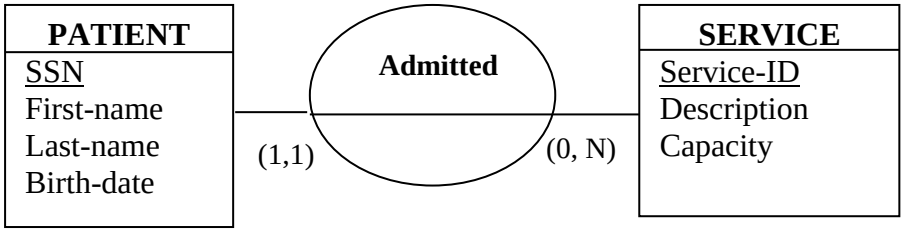
Is converted to the relational relation

Patient (SSN, F-name, L-name, Birth-date)

SSN designates the Social Security Number.

Rule 2: A binary association of type one to many: 1-N disappears with a migration of the key from the parent to the child. The possible pairs of cardinalities in this case are: (1,1)-(1, N); (0,1)- (1, N) or (1,1)-(0, N); (0,1)- (0, N).

Example

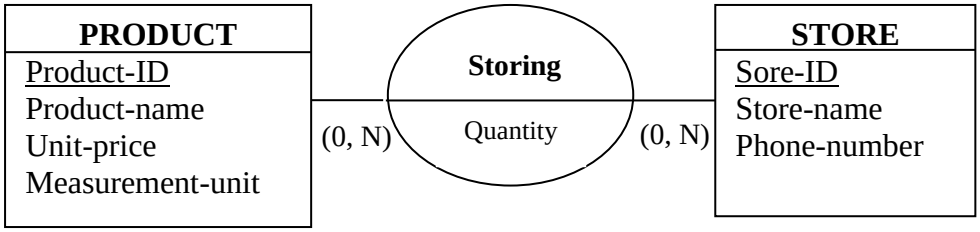


This E/A model is transformed to the following relational model:

Patient (SSN, F-name, L-name, Birth-date, #Service-ID).
Service (Service-ID, Description, capacity)

Rule 3: In a binary association of type many to many, i.e.; (N-N) (*maximal cardinalities are both N*), each entity becomes a relation (*basing on rule 1*) and a novel relation is created. The key of the created relation is composed of the identifiers of the two associated entities. Further, the possible property(ies) of the E/A association are integrated as attributes of the new relation.

Example



By applying the transformation rule 1, we obtain:

Product (#produit, prod-name, Unit-price).
Store (Store-ID, Store-name, Phone-numb).

And the last rule 3 gives the following third relation:

Storing ((#product, Store-ID, Quantity)

Rule 4 : An association of dimension greater than 2, is transformed based on rule 3.

Rule 5 : A reflexive association of type N-N, is transformed basing on rule 3.

Rule 6 : A binary association (1-1) (0-1) disappears with migration of the key of the entity having (0,1) to the entity having (1-1), with a constraint of non-emptiness and uniqueness.

5. Functional dependency (FD) and normalization

Functional Dependency (**FD**) and normalization are two key concepts underlying relational database theory. Before tackling these notions, we illustrate in what follows the need for such fundamental theory.

5.1 Objectives of functional dependency and normalization

Let us consider the **Storage** table below that contains a set of tuples related to different products that are stored in different stores of a company.

#Product	Label	Unit-Price	Quantity	#Store	Address	Volume
100	Pen	15	1000	D1	Annaba	400
200	Pen	15	2000	D2	Guelma	600
300	Eraser	30.50	3000	D4	Guelma	200
400	Board	450.19	700	D1	Annaba	500

As it can be observed in the table, several problems are arising.

- Some information is redundant (***Unit-price, Address***),
- Addition and modification anomalies: When adding/modifying a product, ensure that the price is the same in order to avoid inconsistency. The same issue for stores address (***What is the volume of the store D1?***)
- Deletion anomaly: Deleting the storage of #Product =300 implies the loss of its price and its name. In this case a loss of information is observed.

What to do in order to overcome the previous anomalies?

The goal of normalization is to eliminate storage anomalies by eliminating redundancies.

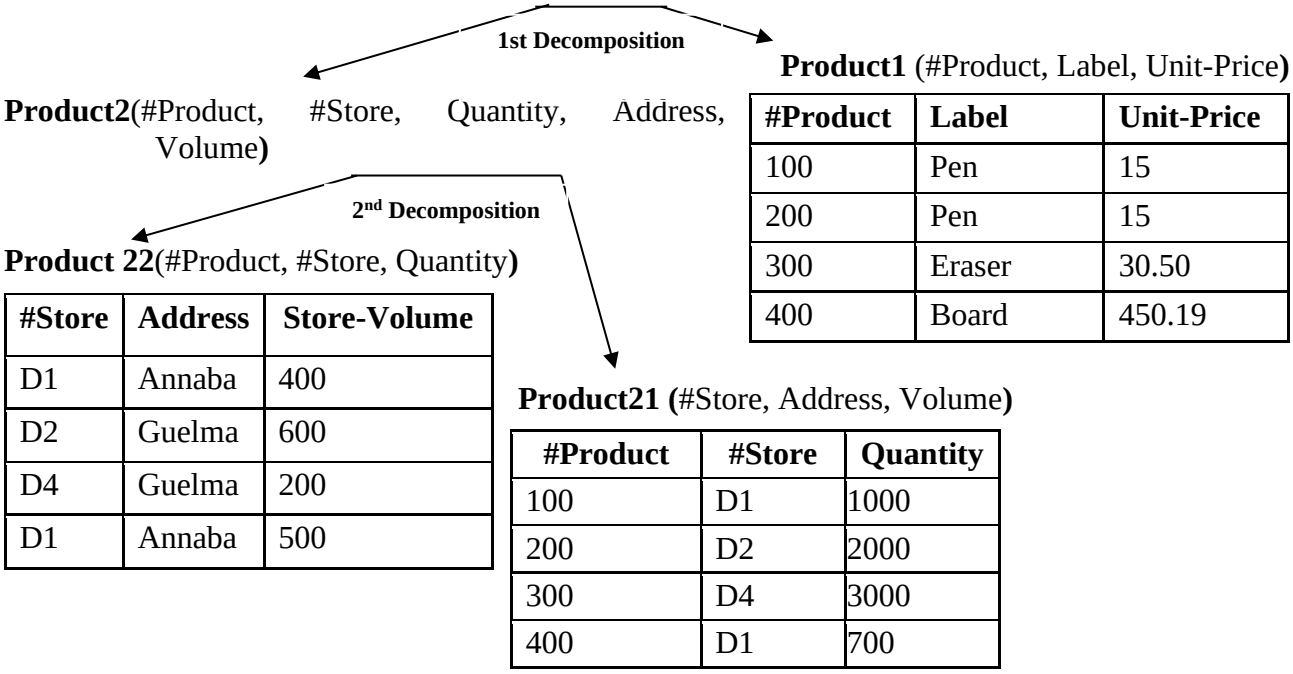
Normalization theory consists of decomposing information into multiple relationships, based on functional dependencies.

The goals of functional dependency and normalization are:

- Limitation of tuple redundancies and, consequently reducing the storage space of data.
- To limit inconsistencies within the data.
- Limiting update issues and improving processing performance.
- Removing NULL values and therefore overcoming difficulties with joins and aggregate functions.

Decomposition of the **Storage** relation

Storage ((#Product, Label, Unit-Price, Quantity, #Store, Address, Volume)



#Product	Label	Unit-Price
100	Pen	15
200	Pen	15
300	Eraser	30.50
400	Board	450.19

#Store	Address	Store-Volume
D1	Annaba	400
D2	Guelma	600
D4	Guelma	200
D1	Annaba	500

#Product	#Store	Quantity
100	D1	1000
200	D2	2000
300	D4	3000
400	D1	700

Figure 2.3 Illustration of the decomposition principle

In what follows, we present the functional dependency principles and the normalization theory is relegated to section 6.

5.2 Functional dependency of a relation

Functional dependency mechanism allows establishing semantic links between attributes or groups of attributes belonging to a relation.

5.2.1 Definitions of functional dependency

Definition 2.2: *We say that an attribute Y (or a list of attributes) functionally depends (**FD**) on another attribute X (or a list of attributes) in a relation R, (denoted $X \longrightarrow Y$) if:*
*given a value of X, it can have **only one value of Y** associated to it in all extensions of R.*

Definition 2.3: *Let R (X, Y, Z) be a relation schema with Z possibly empty.*
*We will say that X **determines** Y in R, or that Y functionally depends on X in R; if:*
$\forall$ the tuples (x,y) and (x',y') of R, $x=x' \Rightarrow y = y'$.
Which means that to each value of X in R corresponds a single value of Y.

Example 1

Considering the following extensions of the relation R, what are the existing FDs?

A	B	C
a1	b1	c1
a1	b1	c2
a2	b2	c2

We can observe that for each value of A (a1, a2, a3) corresponds only an unique value of B (for a1 corresponds always b1, and for a2 corresponds only b2). Hence, the FD :
$A \longrightarrow B$ is established.
The set of the following FDs is satisfied in the previous table.
$B \longrightarrow A$; $A, C \longrightarrow B$; $B, C \longrightarrow A$.
However, the following FD is not satisfied.
$B \not\longrightarrow C$; (*for the same value b1 of B corresponds two distinguished values c1 and c2 of C*).

Other unsatisfied FD in the previous table.
$C \not\longrightarrow B$; $A, B \not\longrightarrow C$.

Example 2

In the relation **Film** bellow, the two FDs are established:
Format $\longrightarrow$ Support and Film-num $\longrightarrow$ Type

Price	Format	Type	Number	Film-num	Support
12	4 :3	Couleur	3	2	VHS
4	16 :9	Noir/Blanc	3	4	DVD
12	16 :9	Couleur	15	56	DVD
35	4 :3	Noir/Blanc	9	12	VHS
12	16 :9	Noir/Blanc	15	12	DVD

Example 3
In the relation **Product (#product, label, unit-price)** of the previous section, we have:
#Product $\longrightarrow$ Label, unit-price. And in the relation **Store (#Store, address, volume)**, we have the following FD: #Store $\longrightarrow$ Address, Volume.
Also, the FD: #product, #Store $\longrightarrow$ quantity is satisfied in the relation:
Storing (#product, #Store, quantity**).**

5.2.2 Properties of FDs (Armstrong's axioms)

Understanding the properties of FDs allows inferring other FDs from existing ones.
Armstrong had elaborated the basic axioms underlying the functional dependency theory and other rules are built basing on this set of axioms.

Axiom 1 (Reflexivity): For each set of attributes X and Y, if $Y \subseteq X \Rightarrow X \rightarrow Y$,
This axiom means that any set of attributes determines itself or a part of itself.

Examples
Student-ID, Student-name→ Student-ID, Student-name
Student-ID, Student-name→ Student-name

Axiom 2 (Augmentation or Expansion): For each set of attributes X and Y:
If $X \rightarrow Y \Rightarrow X, Z \rightarrow Y, Z$;
This rule means that: If X determines Y, then both sets of attributes can be enriched by a third one.

Examples
1) *Course-title, Groupe-number → Prof-name*
   Date, Hour, Course-title, Groupe-number →Date, Hour, Prof-name
2) *Film-title→ Film-genre*
   Film-title, Year → Film-genre, Year

Axiom 3 Transitivity: For each set of attributes X and Y, If: $X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$.

Examples
Course-title, Groupe-number → Prof-name and *Prof-name → Classroom*
Then, by *transitivity we obtain: Course-title, Groupe-number → Classroom*,

Several other rules can be deduced from the previous basic three axioms.

Rule 1 (Union): *If* $X \rightarrow Y$ *et* $X \rightarrow Z \Rightarrow X \rightarrow Y, Z$.

Example
Student-Id → F-name, S-name and Student-Id →Study-year
Then: *Student-Id → F-name, S-name, Study-year*

Rule 2 (Pseudo-transitivity): If $X \rightarrow Y$ *et* $WY \rightarrow Z \Rightarrow XW \rightarrow Z$.

Example
Prof-Num → Grade and Prof-name, Grade → Salary
Then *Prof-Num, Prof-name → Salary*

Rule 3 (Decomposition): *if* $X \rightarrow Y$ *and* $Z \subseteq Y$ *then* $X \rightarrow Z$.
Or: if $X \rightarrow Y, Z$ then $X \rightarrow Y$ et $X \rightarrow Z$.

Example
Film-title → Film-genre, Production-year
Applying the decomposition rule led to the following basic FDs:
Film-title → Film-genre
Film-Title → Production-year

5.2.3 Types of functional dependency

There exist four types of functional dependencies.

a. **Trivial FD: A functional decency $X \rightarrow Y$ is trivial if $Y \subseteq X$.**
   Example
   Label → Label; City-code, City-name → City-name

b. Elementary FD (total): The FD: X→ Y is elementary if: ∀ X'⊆ X, X' doesn't determine Y (*Y does not depend on any part of X*).

Example

#Product → Label is an elementary FD.

#product, #Store → Quantity is an elementary FD.,

However: *#product, label → Unit-Price* is not an elementary FD, because the attribute *Unit-price* depends only on *#product*.

c. *Canonical FD:* *A* Functional Dependency: X→Y is called canonical if its right part is reduced to a single attribute.

Any FD that is not canonical can be transformed by decomposition into a canonical FD.

Example

#Product→Label, Unit-price, is not canonical.

After decomposition, it gives the following two canonical FDs:

#Product →Label and *#Product→ Unit-price.*

d. Direct FD*:* *A* Functional dependency X→ Y is direct if:

(i) It is elementary, and

(ii) Y does not depend on X by transitivity (∄ Z, such that: X→Z and Z →Y).

Example

#Id-employee → F-name, L-name, address is a direct FD.

However: *P1 → P3*, with: *P1 → P2* and *P2 → P3*, is not a direct FD, due to the fact that the FD: *P1 → P3* can be obtained by transitivity.

In the FD: *#Id-employee → F-name, L-name, service-code, address, service-name*, the FD: *#Id-employee → service-name* is not direct, because it exists *Z=service-code* which determines the attribute *service-name*.

5.2.4 Graph of functional dependency

A graph of FDs is an oriented graph that represents all the FDs in a domain.

The vertices of the graph consist of the manipulated attributes and the edges express the existing FDs between attributes. The objectives of representing the FDs in an oriented graph allows an easy visualization of FDs and isolating elementary FDs.

Example: The following set of FDs related to the ***storage*** management area are represented by the corresponding FDs graphs shown in the **table 2.1** below.

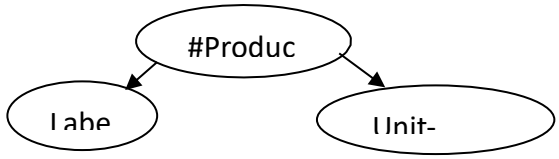
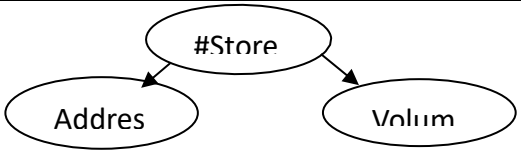
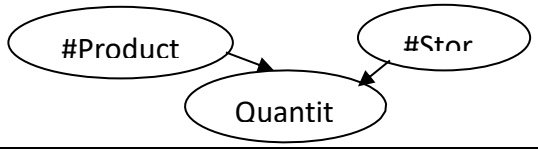
Functional Dependencies	Corresponding Graph of FD
#Product → *Label* ***#Product*** → *Unit-Price*	
#Store → *Address, Volume*	
#Product, #Store → *Quantity*	

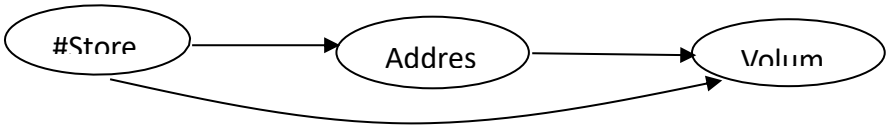
Table 2.1 Graph of the FDs related to the storage management area

5.2.5 Transitive closure of a functional dependencies set

Definition 2.4: *Given a set of elementary functional dependency F, the transitive closure F^+ of F is the set of all elementary functional dependency obtained by adding to F the set of elementary functional dependency inferred from F by transitivity.*

Example 1

F= *{#Store → address, address → volume}*. Thus: $F^+ = F \cup \{\#Store \rightarrow volume\}$.



New FD inferred by transitivity

Figure 2.4 Construction of transitive closure graph

Example 2

F= *{#registration-num → type, type → brand, type → power, #registration-num → color}*
F^+= F ∪ {*#registration-num →brand, #registration_num →power*}.

Objectives of the transitive closure: Constructing the transitive closure of two sets of functional dependencies (FDs) is useful in equivalence checking and database normalization**.**

- **Checking equivalence of two FD sets**

If we have two sets of FDs, F1 and F2, we can compute their closures $F1^+$ and $F2^+$**.**
If $F1^+ = F2^+$**,** then both sets define the same constraints on the relation. This helps in schema normalization when deciding whether two different representations of dependencies are **equivalent**.

- **Minimizing functional dependencies**

By computing the closure of F1∪F2, we can determine whether some FDs in F2 are redundant given F1, and vice versa. This is useful for **reducing redundancy** in database design.

- **Schema decomposition and normalization**

Transitive closure allows checking if a decomposed schema preserves all FDs. The closure helps in verifying whether all original dependencies can still be derived. For example, if we decompose a relation into multiple smaller relations, we check whether their combined FDs still produce the same closure as the original set.

5.3 Relation Keys

The relation's key is a basic concept of the relational model. It refers to a minimal set of attributes of a relation that can determine all the other attributes of this relation.

5.3.1 Key's definition

We give bellow a more formal definition of a relation key.

Definition 2.5*: Let R be a relation having the set of attributes A, and let X be a subset of A.*
X is a key of R if: ∀ Ai ∈ A, X→ Ai.

Example

In the relation *vehicle (#registration-num, brand, type, power, color)*, the attribute *#registration_number* describing the immatriculation of the *vehicle* is a key, because it can determine all the other attributes.
 # registration-num →brand, type, power, color.

5.3.2 Types of relations' keys

a) **Candidate/Primary Key:** A relation R can have multiple keys called candidate keys.
Among the candidate keys, only one is chosen as the **primary key**. It uniquely identifies all records in the table. The primary key is declared with the command: *Primary key*.

Example

Consider the following ***Product*** table in an e-commerce relational database.

Product-ID	Storage reference	Product-Name	Unit-price	Quantity	Category
100	AU110	USB key	100	500	A
101	AK200	Keyboard	200	1000	A
102	CB123	Bicycle	3000	40	C

Table 2.2 An extension of the *Product* table in an e-commerce database

In the previous table, both *Product-ID* and *Storage reference* uniquely identify a product. Thus, *Product-ID* and *Storage reference* are **candidate keys** because each product has a unique ID and a unique *Storage reference*.
A **primary key** is one of the candidate keys chosen to uniquely identify records in the table. In the e-commerce application, the database designer selects ***Product-ID*** as the **primary key** because it is a simple numeric identifier (*often auto-incremented*).

b) **Super key:** Any set of attributes that includes all the attributes of the primary key is called a super key.
Examples: In the **Table 2.2**, *Product-ID,Product-name and Product-ID, storage-reference are super keys.*

c) **Minimal Key:** A subset of attributes X of a relation R is said to be minimal key if :
(i) X is a candidate key;
(ii) Any functional dependency $X \rightarrow A_i$ of R is **elementary**.
Examples: *Product-ID, Product-name, and storage-reference* are minimal keys in the ***Product*** relation of **Table 2.2.**

d) **Foreign Key:** To enable the RDBMS to maintain the consistency of rows in two relationships, the foreign key serves as a link between two relationships in the same database.
Hence, a foreign key allows connecting two tables within a relational database, and ensure the referential integrity of the data. Consequently, only values that must appear in the database are allowed.

Example of foreign keys. Let us consider the following database schema.

Student *(# Num-inscription, F-name, L-name, address, #group},*
Section *(# section, year of study),*
Groupe *(# group, student-number, # section).*

In this example, the *#group* is a foreign key of the relation **Student**, and *#section* is a foreign key of the relation **Group**.
In the relation **Product** of table 2.2, the attribute *category* is a foreign key of the relation.

6. Database normalization

Database normalization is used to correctly design the schema of a relational database. It is a process that helps avoid redundancy, data loss, and data inconsistency issues in the relational model.

Based on the notion of functional dependency, the first three normal forms aim to allow the decomposition of relations without losing information. Each normal form is a progression toward improved relations with less redundancy.

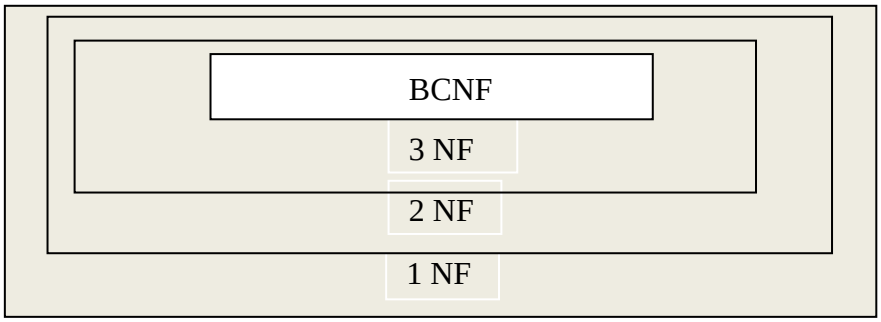


Figure 2.5 The stack articulation of the normal forms

The quality of a relationship is measured by its degree of normalization (1NF, 2NF, 3NF or BCNF).

6.1 1st Normal Form (1NF): A relational relation (*having by definition an identifier*) is in 1NF if each attribute value is atomic (*indivisible*) and mono-valued (*has only one value*).
This definition induces that the attributes of the relation can't have sets or lists of values.

Example 1
In the relation *Teacher (#Registration number, Last name, First name, Grade, Diplomas)*, the attribute *diplomas* can have multiple values. Thus, to allow handling the situation where a teacher can have multiple diplomas, the relation ***Teacher*** is decomposed into the following three different relations.
Teacher (#Registration-Number, Last name, First name, Grade)
Diplomas (#Dip-ID, Title)
Having diploma (#Registration-number, #Dip-ID).

Example 2
In the relation *Book (#book-code, title, pages-number, authors)*, a book can be written by multiple authors. Consequently, the relation **Book** isn't in the 1NF. After decomposition, we obtain two relations in the 1NF.
Book (#book-code, title, pages-number).
Authors (#Mat-author, last name, first name). Write ((#book-code, #Mat-author).

Example 3
Consider the relation *Customer (Customer-ID, Last name, First name, Address).*
Assume that the firm wants to operate some processing on the attribute *address* (*to locate customer for delivery concerns*). In this case, the address is considered as a *not atomic attribute*, and it will be decomposed into its elementary pieces of information.
After decomposition, we obtain the following new *Customer* relation which satisfies the 1NF.
Customer (Customer-ID, Last name, First name, Street, City, State, Postal-code).

6.2 2nd Normal Form: A relational relation *R* is in the second normal form if and only if:

- *R* satisfies the 1st NF.
- It has no partial dependencies (*attributes that do not belong to a key do not depend on any part of that key*). Hence, all the FDs between the key and the other attributes of *R*are **elementary**.

Example: Consider the following relation ***Stock***.
Stock *(#Product-ID, #Store, Product-label, quantity).*
With the FD: #Product-ID, #Store → *Product-label* and #Product-ID, #Store → Quantity.

The ***Stock*** relation is not in the 2nd NF, because the attribute *Product-label* depends on a part of the key. In fact, the *Product-label* can be determined only by the *#Product-ID.*
Thus, the FD: *#Product-ID, #Store → Product-label* is not elementary.

The decomposition of the previous relation ***Stock*** leads to two relations which are in the 2NF.
Product (#Product-ID, Product-label).
Stock (#Product-ID, #store, quantity).

A relation in the 2NF may also contain redundant information. Therefore, the 2NF does not avoid all redundancies in the database.

6.3 3rd Normal Form
A relational relation *R* is in 3rd Normal Form (3NF) if:

- It is already in 2nd Normal Form.
- It has no transitive dependencies. Hence, all the FDs are **direct**. Any attributes not belonging to the key do not depend on a non-key attribute. In other words, non-key attributes should not depend on other non-key attributes.

Example 1
Consider the following relation *Teacher*:
Teacher (#Teacher-ID, Last-name, First-name, Grade, Weekly-workload).
With the FD: *Grade → Weekly-workload.*
The relation *Teacher* is not in 3rd NF because a non-key attribute (*Grade*) determines another non-key attribute (*Weekly-workload).*
The previous relation must be split into two separate relations satisfying the 3rd NF:
Teacher (#Teacher-ID, Last-name, First-name, Grade)
Workload (Grade, Weekly-workload).

Example 2

Let us consider the extension schema of the relation ***Vehicle*** given bellow.

#Registration-num	Brand	Type	Power	Color
00215-123-24	Peugeot	406	7	Blue
74851-118-23	Fiat	500	6	Gray
45874-119-16	Renault	25	5	White
00123-100-16	Peugeot	406	7	Black
56123-124-31	Peugeot	3008	8	Black

As it can be observed in the table expressing the extension of the relation ***Vehicle*** *(#Registration-num, Brand, Type, Power, Color),* the attribute **Type** can determine the *brand* and the *Power* of the vehicle*. More formally: Type → Brand, Power. Consequently, the relation* ***Vehicle*** *does not satisfy the 3rd normal form.*

After its decomposition, we obtain the two relations which are in 3NF.
Vehicle *(#Registration-num, Type, Color),* and ***Model*** *(Type, Brand, Power).*

The 3NF is also insufficient for avoiding all the anomalies and redundancies. This can occur when a relation admits multiple candidate keys. In fact, even though the relation is in 3NF, other cases of anomalies can be observed.

6.4 Boyce and Codd Normal Form: A relational relation *R* is in Boyce-Codd Normal Form (BCNF) if and only if:

- It is in 3NF.

- For all functional dependencies, the left side is a key. Thus, its only elementary FDs are those where a key determines an attribute: X→ Y ⇒ X is a key.

Example 1

Consider the relation ***Localization*** *(City, Street, Postal-code)* which is in 3NF.

City	Street	Postal-code
Boumahra Ahmed	Rue 1er Novembre	24005
Hamam Debagh	Rue des amendiers	24007
Boumahra Ahmed	Chorfa Ahmed	24005
Hamam Debagh	Rue des Chouhada	24017
Boumahra Ahmed	Rue des amendiers	24007

Let's assume the relation contains the FDs:
City, Street → Postal-code and *Postal-code → City (a non-key attribute determines another attribute).* Thus, the relation ***Localization*** is not in the BCNF. Although this relation is in 3NF, it contains redundancies.
The decomposition of the localization relation gives the following two separate relations.
R1 (Postal-code, city) and *R2 (Street, Postal-code).*
The FD: *City, Street → Postal-code* is not preserved.

Example 2

Let the relation **Person** (#SSN, #Country, Name, Region) with the following FDs in this relation: SSN, Country → Name, SSN, Country→ Region; Region→ Country.
The last FD is not derived from a key and expresses a FD where a non-key attribute determines an attribute belonging to a key. Hence, the relation Person is in 3NF, but not in BCNF (*BCNF stipulates that all FDs are derived from a key*).
To have a relational schema in BCNF, we must decompose **Person**, as bellows:
Person *(#SSN, Name, Country)* and ***Region*** *(#Region, Country).*

Theorem: *Any relational relation has a BCNF decomposition without loss of information.*

7. Database schema and integrity constraints

Here after, the database schema is described and the underlying set of constraints related to a particular database are discussed.

7.1 Database schema

A database schema is a structural description of the database. It expresses how the data is organized in the database, the relationships between tables, and the database constraints, but it does not contain the data itself.

Definition 2.6: *The schema of a relational database is composed by the set of relation schemas that make up the database, as well as the fields included in each table.*

Definition 2.7: *A database schema is the logical structure that defines the organization of data within the database. It describes:*

- *Tables and their columns;*
- *Data types of each column;*
- *Relationships between tables (primary keys, foreign keys);*
- *Constraints (uniqueness, non-nullity, etc.) ;*
- *Views, indexes, and stored procedures (in some cases)*

Example

Consider the library book loan management. The database schema is composed of the following relations' schema.

1. *Authors (author_id, name, birth_date);*
2. *Books (book_id, title, genre, published_year, author_id)*
3. *Members (member_id, full_name, email, phone, membership_date)*
4. *Borrowings (borrowing_id, member_id, book_id, borrow_date, return_date, status)*

The Primary and foreign keys of the database schema are given bellow.
Primary Keys: *author_id, book_id, member_id, borrowing_id*
Foreign Keys:
Books.author_id → Authors. author_id
Borrowings.member_id → Members.member_id
Borrowings.book_id → Books.book_id

7.2 Integrity constraints

The data manipulated in the database is not independent, but obeys particular conditions and constraints which ensure the accuracy, consistency, and validity of data. These constraints prevent invalid data entry and maintain data relationships.

Constraints imposed to databases refers to a set of semantic rules applied to attributes and relations to maintain its consistency. Hence, during the database updates, the DBMS must ensure overall data consistency by respecting the previously specified constraints.

The following four types of constraints are distinguished.

a) Domain's constraints: ensure that an attribute's values fall within a specified domain (*data type, range, format*). For example, the *marks* in the *student* table should only contain values between 0 and 20.

b) Key unicity constraints: a relationship must have a primary key whose value is unique and non-null. For example, the *Product-ID* of **Table 2.2** is a primary key having unique and distinguished values for each tuple of the ***Product*** relation.

c) Referential constraints: these constraints maintain consistency between **foreign keys** and **primary keys** in related tables. A **foreign key** must either reference an existing value in the referenced table or be NULL. For example, if a **Product** table has category attribute as a foreign key, that category must exist in the **category** table.

d) Non nullity constraint: ensures that a particular attribute can't have NULL values. For example, the value of students' numbers of a group can't be NULL, and ordered quantity of a product can't be Null.

8. Conclusion

The relational model remains the most efficient model used by companies to handle their data. It constitutes the cornerstone of modern database systems and provides a structured and logical approach to data organization.

This chapter has outlined the fundamental definitions and concepts that form the basis of the relational model. The transition from the Entity-Association model to the relational model highlights the importance of conceptual design in database development.

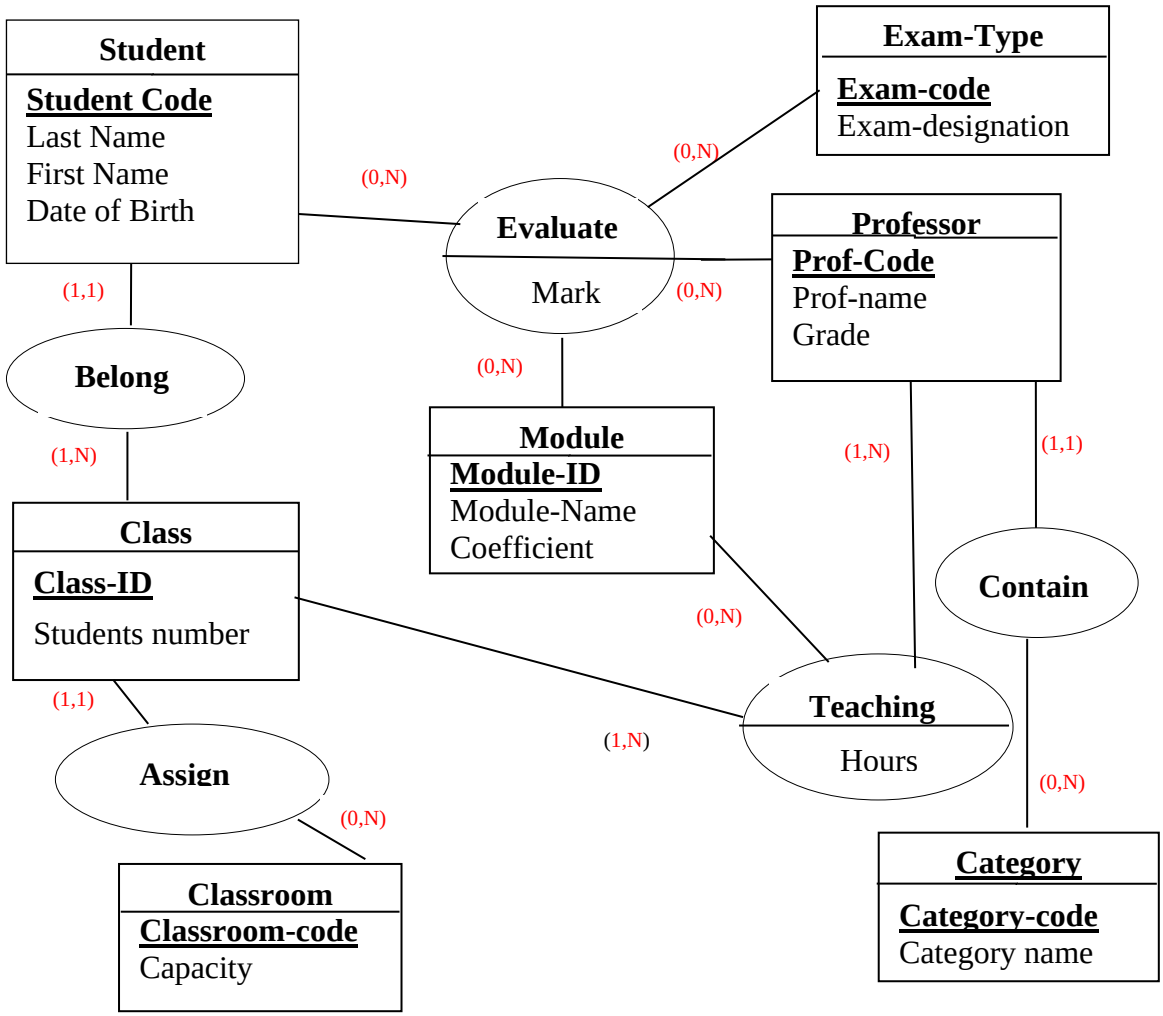
A key aspect of relational databases is ensuring data consistency and integrity through functional dependencies and normalization. By defining functional dependencies, exploring their properties, and establishing relation keys, we ensure efficient data structuring. The process of normalization, from the first normal form to Boyce-Codd Normal Form, plays a crucial role in minimizing redundancy and maintaining data integrity.

Finally, database schema design and integrity constraints provide essential guidelines for ensuring reliable and consistent data storage. Together, these principles form a robust framework for designing, managing, and optimizing relational databases, reinforcing their role as a fundamental technology in data management.

The next chapter is dedicated to the presentation of the relational algebra used for manipulating relational tables.

Series of exercises No. 2

Exercise 1 : Let us consider the following Entity/Association model relating to the management of an educational establishment.



1. Transform the E/A model to a relational model?
2. Give a possible extension of the **Teaching** relation?
3. What is the key of the **Evaluate** relation?

Exercise 2: Mastery of Functional Dependencies

1. Consider the relation R (A, B, C, D, E, F) in which a set of functional dependencies is defined on the left column of the table. Complete the empty boxes in the table on the right.

A, B → C D → C D → E C, E → F E → A	**A**	**B**	**C**	**D**	**E**	**F**
		1		*110*		*54*
	x	*2*	*j*	*100*	*n*	*52*
	w	*1*	*i*	*110*	*m*	
		2		*100*		*52*

2. Let the relation ***Film (Title, Year, Length, filmType, StudioName, starName)*** with its extension illustrated by the set of tuples of the following table.

title	year	length	filmType	studioName	starName
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Esteves
Wayne's World	1992	95	color	Paramount	Dana Carvey
Wayne's World	1992	95	color	Paramount	Mike Meyers

a. Give a trivial FD, a canonical FD, an elementary FD, and a direct FD.
b. Identify all the possible elementary FDs and formulate them in canonical form.
c. Draw the graph of the FDs.

Exercise 3

a. Consider the relation ***R (A, B, C)*** with the set of FDs: {A → B; B → C}.
For example, R could be the relation *FILM (No_exploitation, Title, Director)*.
1) What is the primary key of R?
2) In what normal form is the relation *R*?
3) Is the following extension of the relation R' an extension of R?

R'	A	B	C
	A1	B1	C1
	A2	B1	C2
	A3	B2	C1
	A4	B3	C3

4) By using the relation ***R'***, find an extension ***R''*** that conforms to ***R***.
5) Propose a decomposition in the 3NF of ***R*** without loss of information.

Exercise 4

The following **ORDER** relationship describes orders placed by customers and lists the ordered products and the associated quantities.

ORDER (Order-ID, Order-date, Customer-ID, Cust-address, Product-ID, Price, Quantity)

Order-ID	**Order-date**	**Customer-code**	**Cust-address**	**Product-ID**	**Price**	**Quantity**
C217	26/03/2025	CUS423	Berrahal-Annaba	P983	200	575
C902	14/10/2025	CUS611	Sidi-rached-Setif	P120	650	575
C465	10/10/2024	CUS423	Berrahal-annaba	P482	650	88
C510	14/10/2025	CUS064	Nouvelle ville-Annaba	P983	200	204
C510	14/10/2025	CUS064	Nouvelle ville-Annaba	P005	4700	196

In the table, lines of each order describe that the product is characterized by its price and a specific quantity is given for each ordered product.

1. What is the key of the relation ***Order***?
2. What is the normal form of the ***Order*** relation?
3. Convert the relation ***Order*** to 3NF if applicable?

Exercise 5 :

Let ***R1 (A, B, C, D, E, F)*** be a relation having the following set of functional dependencies:
{A, B → C; A → D; A, B → E; A, B → F; B → C ; D → E ; D → F}.

1. Draw the functional dependency graph associated to ***R1***.

2. Give the graph (set) of minimal functional dependency. What is the key of ***R1***?

3. What is the normal form of the relation ***R1***?

4. We decompose the relation ***R1*** in two separate relations ***R11*** et ***R12***, such that:
   R11(A, B, D, E, F) *and* ***R12 (B, C)***.
   What are the normal forms of ***R11*** and ***R12***?
   Propose a decomposition of ***R11*** without loss of information.

Chapter III: The Relational Algebra

1. Introduction

The relational model, discussed in the previous chapter, allows organizing data in the form of interconnected relations, each of which represented in a tabular format. Thus, relations constitute the fundamental and unique components of the relational model. Such relations are expressed through: (i) an abstract schema (*a set of field names*), (ii) an extension of the relation (*a set of tuples*). Basing only on the database schema, composed of a set of relations' schemas, a relevant issue consists to perform operations on these relations in order to deduce new relations from existing ones.

The relational algebra, provides a satisfactory answer to this concern.

This chapter is dedicated to the presentation of the relational algebra. It starts by defining the relational algebra. Then it discusses the motivations having led to the elaboration of this theory. After that, it exposes the basic unary operations (*union and intersection*). Finally, the set operations are deeply presented. For each operator, a real-word illustration is given in order to consolidate the considered operator's usage.

2. Definition

Invented by E. Codd, the relational algebra is a formal system for manipulating and querying data in relational databases. It consists of a collection of formal operations acting on relations to produce new relations as results. Hence, it allows the specification of the operations to be performed in order to calculate the result of a query.

Definition 3.1*: The relational algebra provides a set of operations that take one or more relations (tables) as input and produce a new relation as output, without modifying the original data.*

The relational algebra is an operational language for expressing queries as a succession of operations on relations. These operations are mathematically precise and serve as the foundation for SQL query optimization.

3. Motivations for using relational algebra

The following motivations reinforce the reasons that led to the development of the relational algebra and its intensive usage in databases field.

- **Data manipulation:** The relational algebra provides standardized set of operations (*selection, projection, join, union, difference, …etc.*) that facilitate data manipulation in a consistent and easy manner.
- **Abstraction of physical details:** Relational algebra allows users to focus on logical operations rather than the physical details of data storage.
- **Declarative language:** Relational algebra is a declarative language allowing users to specify only what are their data needs, without worrying about how to get them. Therefore, a clear separation between query specification and the underlying implementation is performed.
- **Solid theory:** Relational algebra is based on a solid theoretical foundation, articulated on the set and predicate theories. Thus, the defined operations have well-defined properties. This helps ensuring the consistency and reliability of data operations.
- **Structured querying:** Relational algebra-based queries allow complex queries specification in a structured manner. This makes queries easier to understand and maintain.

The relational algebra offers a formal and powerful tool to manipulate data in a consistent, and efficient way. In the following two sections 3 and 4, the operations proposed by the relational algebra are exposed and illustrated. For each operator, we give its definition and notation, followed by the graphical representation and we illustrate its usage with a real-world example.

4. Unary operators

This operations' type takes as input an initial relation and constructs a new relation as result. We distinguish two operators: the selection and the projection.

4.1 Selection

This operator consists of creating from and initial relation R1(A_1, A_2, ..., A_n), a new relation R2(A_1, A_2, ..., A_n), having the same schema, but whose tuples are those of R1 satisfying a particular condition C. The condition C is in the form:

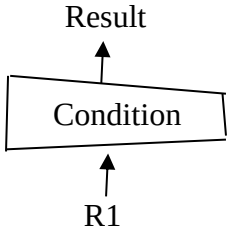
<Attribute> <Operator> <Value>,

Where the ***Operator*** belongs to the comparison set : $\{=, <, \leq, \geq, >, \neq\}$.

Operator notation: The following notations can be used to express the selection operator.

- $\sigma_{condition}$ (R1),
- R1 [Condition]
- RESTRICT (R1, Condition)

Schematic representation



Example

Let ***Customer*** be a relational relation having the following tuples.

Customer-ID	First-name	Last-name	Age
100	Ahmed	Rahmani	52
200	Mohamed	Oussaad	30
207	Selma	Driss Hamed	47
215	Amina	Rahmani	21

Table 3.1 Example of an initial relation *Customer*

The selection query: $\sigma_{age>30}$ **(Customer)** applied to the initial relation ***Customer*** produces a relation having the following two tuples that satisfying the selection condition (*age >30*).

Customer-ID	First-Name	Last-Name	Age
100	Ahmed	Rahmani	52
207	Selma	Drissi Hamed	47

4.2 Projection

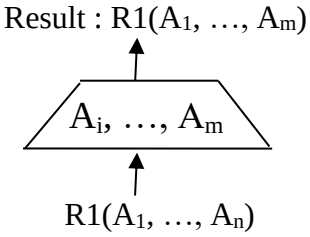
Instead of selecting a subset of tuples, the projection operator focuses on a subset of attributes. More formally:

The projection of a relation R1(A_1, …, A_n) onto the attributes (A_i, …, A_m), (*with m<n*), consists of creating a relation R2(A_i, …, A_m) having the only attributes A_i, …, A_m obtained by removing from R1 the attributes not mentioned as operands and by eliminating the potential duplicate tuples that risk appearing in the resulting table R2.

Operator notation: The following notations can be used to express the projection operator.

- $\Pi_{A1, A2\ldots Am}(R1)$
- R1 [Ai, Aj... Am]
- PROJECT (R1, A1, A2, …, Am)

Schematic representation



Example

The projection query: Π**Customer-ID, Age** (***Customer***) refers to the projection of the relation ***Custome***r on the two attributes ***Customer-ID*** and ***Age***. The resulting table is shown below.

Customer-ID	**Age**
100	52
200	30
207	47
215	21

5. Set operators

In this category, a third relation is constructed from two input relations.

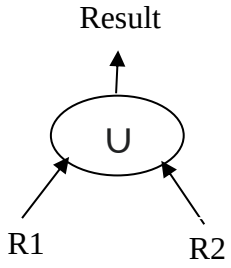
5.1 Union

The union of two relations R1 and R2 having the same schema is a new relation R. The tuples of the new relation R belong to either R1 or R2.

Operator notation: The following notations can be used to express the union operator.

- R= R1 ∪ R2
- UNION (R1, R2)
- APPEND (R1, R2)

Schematic representation



Example

Let *Product-Oran* and *Product-Annaba* be two relations managed by a company. These two relations have the same schema and contain goods stored in *Oran* and *Annaba* stores, respectively. The tuples of the relations are illustrated in the following tables.

Product-Oran

Product-ID	Product-Name	Price
100	Printer	40000
200	Screen	20000
300	Mouse	500
400	Keyboards	1000

Product-Annaba

Product-ID	Product-Name	Price
500	USB key	1500
200	Screen	20000
600	CPU	1500
300	Mouse	500

Table 3.2 Illustration of the tuples of the two stores

By applying the ***Union*** operator to the two previous relations, a new relation, named "***All-Product"*** is obtained. The tuples of the new relation are shown below.

All-Product = Product-Oran ∪ Product-Annaba

Product-ID	Product-Name	Price
100	Printer	40000
200	Screen	20000
300	Mouse	500
400	Keyboards	1000
500	USB Key	1500
600	CPU	1500

We notice that the redundant tuples occur once in the resulting relation.

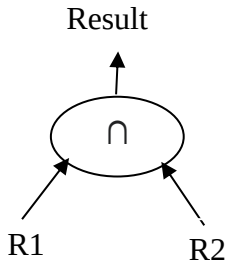
5.2 Intersection

The intersection of two relations R1 and R2 having the same schema return a new relation R containing the tuples that are present in both relations.

Operator notation: The following notations can be used to express the intersection operator.

- R= R1 ∩ R2
- INTERSECT (R1, R2)
- AND (R1, R2)

Schematic representation



Example

The application of the operator ***Intersection*** to the two store relations of table 3.2 gives a new relation, named "***Common-Product***" containing the products belonging to both stores (*Oran* and Annaba) at the same time. The tuples of the resulting table "***Common-Product***" are shown below.

Common-Product = ***Product-Oran*** ∩ ***Product-Annaba***

Product-ID	Product-Name	Price
200	Screen	20000
300	Mouse	500

5.3 Difference

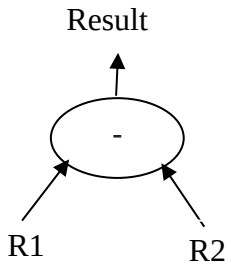
This operator acts on two relations R1 and R2 with the same schema and constructs a new relation R3 with the same schema having all tuples that are in R1 but not in R2.

Note: This operation is non-commutative ($R1-R2 \neq R2-R1$).

Operator notation: The following notations can be used to express the difference operator.

- R= R1 - R2
- DIFFERENCE (R1, R2)
- REMOVE (R1, R2)
- MINUS (R1, R2)

Schematic representation



Example

The application of the operator ***Difference*** to the two store relations of table 3.2 gives a new relation, named "***Exist-Oran***" that contains all the products that are stored in *Oran* store but don't exist in *Annaba* one. The resulting tuples are given bellow.

Exist-Oran = ***Product-Oran*** - ***Product-Annaba***

Product-ID	Product-Name	Price
100	Printer	40000
400	Keyboards	1000

It should be emphasized that the difference operation is non-commutative.

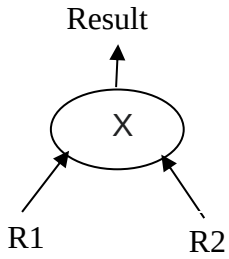
5.4 Cartesian product

The Cartesian product of two relations R1 and R2, which do not necessarily have the same schema, is a relation R3. The schema of the resulting relation R3 consists of the concatenation of the attributes of R1 and R2, and whose tuples consist of all possible combinations between the tuples of R1 and the tuples of R2.

Operator notation: The following notations express the Cartesian product operator.

- R= R1 x R2
- PRODUCT (R1, R2)
- TIMES (R1, R2)

Schematic representation



Example

Consider the relations: ***Country (Name, Capital, Currency)*** and ***Currency (code, label)*** having the following occurrences.

Country

Name	Capital	Currency
Algeria	Algiers	5
Tunisia	Tunis	6

Currency

Currency	Description
5	Algerian Dinar
6	Tunisian Dinar
10	Euro

The Cartesian product operator applied to the two previous relations, gives as result a new relation, named "***All-Currency***". The tuples of the new relation are shown below.

All-Currency = Country X Currency

Name	Capital	Currency-code	Currency-ID	Description
Algeria	Algiers	5	5	Algerian Dinar
Algeria	Algiers	5	6	Tunisian Dinar
Algeria	Algiers	5	10	Euro
Tunisia	Tunis	6	5	Algerian Dinar
Tunisia	Tunis	6	6	Tunisian Dinar
Tunisia	Tunis	6	10	Euro

NB: *If both relations R1 and R2 have an attribute with the same name, we rename this attribute or specify the name of the relation to which it belongs (e.g., R1.A, R2.A).*

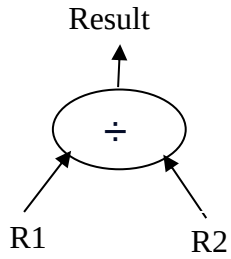
5.5 Division

It concerns two relations R1 and R2, such that the schema of R2 is strictly included in that of R1. The generated relation R3 will have the schema of R1 minus the attributes of R2. The tuples of the resulting relation R3 are formed from all the tuples that, when concatenated with each tuple of R2, always yield a tuple of R1.
This operation allows finding which tuples in a relation are associated with all the tuples in another relation. Consequently, it answers queries of the form: "***For all values of X, find Y?***"
For example: *What are the addresses of the customers who purchased all the products?*
Relation R2 cannot be empty. If R1 is empty, the resulting relation of the division is empty.

Operator notation: The division operator is noted as follows.

- R= R1 / R2
- DIVISION (R1, R2)

Schematic representation



Example

Let the two following relations: ***R(A,B)*** and ***S(A)***, with their respective extensions. The division operation: T(B) = R(A,B) ÷ S(A) gives the tuples depicted in the table at the right.

R(A,B)

A	B
a1	b1
a1	b2
a2	b2
a1	b3
a2	b1

S(A)

A
a1
a2

R(A,B) ÷ S(A)

B
b1
b2

5.6 Join operator

The join operator is used to combine two relations (*tables*) based on a related attribute (*typically a common key or field*). It produces a new relation that includes combinations of tuples from the input relations that satisfy a specific condition.

The following three types of the join operator are distinguished.

5.6.1 Theta-join (θ-join)

The θ-join of two relations ***R1*** and ***R2*** according to a condition ***C*** is a new relation ***R3*** having as a schema the concatenation of the attributes of R1 and R2, and the tuples are those of the Cartesian product between R1 and R2 satisfying the condition C.

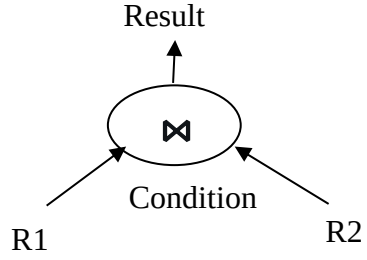
The condition ***C*** is of the form: ***R1.ai θ R2.bj***, where θ is one of the operators (=, <, ≤, ≥, >, ≠). If the operator is the equality "=", the θ-join is called an **equi-join**, and if it is the inequality "≠" one, the θ-join is called an **Inequi-join**.

The θ-join is equivalent to a Cartesian product followed by a selection operation.

Operator notation: The join operator can be expressed as follows.

- $R1 \bowtie_{C} R2$
- JOIN (R1, R2, CONDITION)

Schematic representation



Example 1

Consider the two following relations:

Employee *(Name-Emp, Salary-Emp)* and ***Manager*** *(Name-Man, Salary-Man),* with their respective extensions.

Employee

Name-Emp	Salary-Emp
Saadouni	20 000
Mansouri	10 000
Benmohamed	6 000

Manager

Name-Man	Salary-Man
Drarenia	25 000
Touhami	12 000
Rahmouni	20 000

Now, let's consider the θ-join operation:

Superior*=Employee* ⋈ *Manager* on condition *C: Salary-Emp > Salary-Man.*

This θ-join allows us to answer the question:

Who are the employees who are paid more than their manager?

Only one tuple satisfies the previous condition, and the resulting new relation ***Superior*** is shown hereafter.

Name-Emp	Salary-Emp	Name-Man	Salary-Man
Saadouni	20 000	Touhami	12 000

Example 2: Equi-join on the salary attribute.

We want to know who are the employees earning the same salary as their bosses?
To answer the question, we use the equi-join operator on the salary attribute.

The equi-join query is formulated as follows:

Equal-Salary *=Employee* ⋈ *Manager*

on the condition *Salary-Emp=Salary-Man.* This query gives as result the following tuple.

Equal-Salary

Name-Emp	Salary-Emp	Name-Man	Salary-Man
Saadouni	20 000	Rahmouni	20 000

5.6.2 Natural-Join

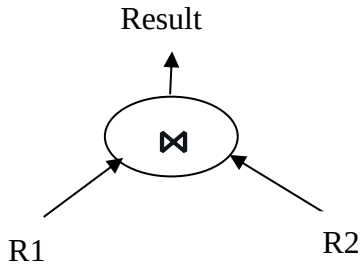
The natural join of two relations ***R1*** and ***R2***, which do not necessarily have the same schema, refers to an equi-join operation on all the common attributes, while removing multiple occurrences of these common attributes.

The attributes of the new relation ***R3*** are obtained by concatenating the attributes of ***R1*** and ***R2***, and its tuples are those of ***R1*** and ***R2*** having the same values for the same attributes (*respecting an equi-join between common attributes*). These common attributes are not duplicated in relation ***R3***, but merged into a single column. Further, there is no need to specify the condition in a natural join, because the join condition always remains implicit.

Operator notation: The natural join operator can be expressed as follows.

- R1 ⋈ R2
- JOIN (R1, R2)

Schematic representation



Example 1

Consider the two following relations: ***Product*** and ***Project*** and their respective extensions.

Product-ID	Project-ID
100	1000
100	2000
200	3000
600	5000

Project

Product

Product-ID	Supplier
100	A
100	B
200	C
700	C

Assume that we want to know each product is used in which project.

To answer this user's need, we formulate a natural join between the two previous relations ***Product and Project***. The resulting relation, ***Product-in-Project*** is specified as follows.

Product-in-Project= Product ⋈ Project

The result of the previous query gives the following tuples.

Product-in-Project

Product-ID	Supplier	Project-ID
100	A	1000
100	A	2000
100	B	1000
100	B	2000
200	C	3000

5.6.3 Outer join (or External join)

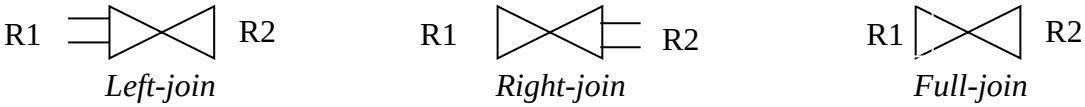
The outer join operation between two relations ***R1*** and ***R2*** with any schema is a new relation ***R3*** whose schema is composed of the concatenation of both the attributes of ***R1*** and those of ***R2***, while representing the common attributes (*with the same name*) only once.
The tuples in the resulting relation ***R3*** are obtained with a natural join between R1 and R2, plus those of the two initial relations ***R1*** and ***R2*** not participating in the join by representing those of the other relation with **NULL** values.

According to the values of the tuples contained into the two relations participating to the outer join operation, three types of outer join are distinguished: Left-join, Right-join and Full-outer join.

Operator notation: The outer-join operators are represented as follows.

- LEFT OUTER JOIN (or just LEFT JOIN)
- RIGHT OUTER JOIN (or just RIGHT JOIN)
- FULL OUTER JOIN (or just FULL JOIN)

Schematic representation



Example 1

Consider the two following tables A and B.

Store A

Product-Label	***Product-ID***
X1	*1*
X2	*2*
X3	*3*

Store B

Product-ID	***Product-Quantity***
1	***Y1***
2	***Y2***
4	***Y3***

The left-join between the two relations gives the following tuples.

Left-Join (Store A, Store B)

Product-ID	***Product-Label***	***Product-Quantity***
1	*X1*	*Y1*
2	*X2*	*Y2*
3	*X3*	***NULL***

The right-join between the two relations gives the following tuples.

Right-Join (Store A, Store B)

Product-ID	***Product-Label***	***Product-Quantity***
1	*X1*	*Y1*
2	*X2*	*Y2*
4	***NULL***	*Y3*

The tuples of the full-join operator are illustrated in the following table.

Full-Join (Store A, Store B)

Product-ID	***Product-Label***	***Product-Quantity***
1	*X1*	*Y1*
2	*X2*	*Y2*
3	*X3*	***NULL***
4	***NULL***	***Y3***

6. Conclusion

In this chapter, we provided a detailed presentation and illustration of relational algebra. Introduced by E. Codd, this formal system is designed for querying and manipulating relational tables through a set of mathematically rigorous operations. Independent of the physical storage of data, relational algebra serves as a foundation for the query optimization techniques employed by SQL.

Relational algebra operates on one or more input relations and produces new relations as output. It includes a set of well-defined operations—such as selection, projection, union, intersection, difference, Cartesian product, division, and various types of joins (including theta joins, natural joins, and outer joins). These operations are explained and illustrated through real-world examples to highlight their practical usefulness in extracting meaningful information from relational databases.

In the next chapter, we will address the purely practical aspect, focusing on the SQL language for creating the database and manipulating data.

Series of exercises No. 3

Exercise 1

Let us consider the following relational model of a school management database.

Student (Student ID, Name, Address)
Course (Course ID, Day, Time)
Assign (Course ID, Room ID)
Evaluate (Course ID, Student ID, Mark)

A possible instantiation of the DB is illustrated by the following extensions.

Student

Student ID	Name	Address
100	Mohamed	Guelma
200	Said	Skikda
300	Rana	Annaba

Course

Course ID	Day	Time
Archi	Sunday	09H
ASD	Monday	14H
DB	Wednes.	08H
DB	Tuesday	11H

Assign

Course ID	Room ID
Archi	L3.2
ASD	E8.9
DB	Amphi 3

Evaluate

Course ID	Student ID	Mark
Archi	100	12
Archi	300	13
ASD	100	09
ASD	200	11
ASD	300	06
DB	200	03
DB	300	16

1. Give the results of the following projections.
   Q1 = $\pi_{\text{Course ID}}$ (Course)
   Q2 = $\pi_{\text{Student ID, Address}}$ (Student)
2. Give the result Q3 of the restriction $\sigma_{\text{Course ID = 'ASD'}}$ (Evaluate).
3. Express the following queries in the relational algebra (RA).

   Q4: List of students with their addresses.
   Q5: What is the teaching time for the DB course?
   Q6: The identifiers of students having obtained more than the average in the DB course.

Exercise 2

Consider the database schema for a company's employees management.

EMPLOYEE (E-Num, E-Name, Profession, Hire-Date, Salary, Commission, #Dep-Num)
DEPARTMENT (#Dep-Num, Dep-Name, Director, City-D)

Formulate the expressions in Relational Algebra that meet the following requirements:

Q1: List of engineers (*Number, Name*) hired before 01/01/2020, along with their commission.

Q2: Give all tuples of EMPLOYEE.

Q3: Give the hire dates of the technicians.

Q4: The name and profession of employee number 10.

Q5: Give the employees' professions (*after eliminating redundant tuples*).

Exercise 3

Let R and S be two relations in which the attributes A, B, and C are defined on the domains of alphabetic letters.

R

A	B
a	b
a	f
c	b
d	e

S

B	C
b	c
e	a
b	d
g	b

Give the results of the following queries.

1. R ⋈ S (natural join)
2. $\sigma_{A=C}$ ($\rho_{B'/B}$ (R x S) (*Equi-join; ρ denotes the renaming operator and "x" the Cartesian product*)
3. R ⋉ S = π_R (R ⋈ S) (Semi-join)
4. Are the following relations true?

   $\pi_{A, B}$ (R ⋈ S) = R and $\pi_{B, C}$ (R ⋈ S) = S

Exercise 4

Consider the database schema from **Exercise 2**.

Express in the relational algebra the queries meeting the following requirements:

Q1: Give the names of the employees and their departments.

Q2: Give the employees numbers working at ANNABA.

Q3: Give the names of the managers of Departments 1 and 3.

Q4: Give the names of the employees working in a department with at least one engineer.

Exercise 4

Let us consider the two relations STUDENT and RESIDENT which designate, respectively, the students of the faculty and the residents of the university residences.

STU

RESIDENT

ID	Residence	Room
200	Springer	A20
500	2000 beds	B20
400	Springer	C15

Express the following queries in the relational algebra

1. Q1: "*What are the numbers of the residents who are not students?*"
2. Q2: "*The list of students (numbers, name), with the residence and the room number of each one?*"
3. Q3: "*The names of the students who reside in the "Springer" residence, with their room numbers?*"

Chapter IV: The Structured Query Language (SQL)

1. Introduction

In the first chapter, the Data Base Management System (DBMS) was exposed as an inevitable software component useful for managing data stored in databases (***See § 5 in page 7***). In fact, this tool ensures the junction between the users' view and the physical description materialized by the database files. On the other hand, the second chapter has introduced the relational model and a set of adequate techniques (functional dependencies and normalization) for enabling the design of database schemas that are integrated, consistent and having minimal redundancies. Once the DB schema is designed and normalized, the DB designer will focus on its actual implementation on a physical medium. This step requires deploying the functionalities of the DBMS and its associated language SQL (**S**tructured **Q**uery **L**anguage). Such a software tool allows creating the DB structure, manipulating and managing data, as well as requesting data from the conceived DB.

This chapter focuses on the SQL language and its associated features used to formulate user queries. It begins with an introduction to commands for defining the database structure and manipulating data. Next, it covers the commands used to query data and retrieve relevant information. Finally, a range of useful aggregation functions is described and illustrated through practical examples.

We start by giving a brief description of SQL.

2. SQL Description

Defined by ANSI (*American National Standards Institute*) and ISO (*International Standards Organization*), SQL is a standard computer language that ensures communication with relational databases management systems (RDBMSs)**.** It offers a declarative language that allows users to query a database.

A more precise definition is given below.

Definition 4.1*: SQL is a standard declarative programming language that allows users managing and manipulating relational databases without worrying about the internal (physical) representation of the data, its location, access paths or the necessary algorithms.*

The success of the SQL language is mainly due to its **simplicity** and the fact that it relies on the **conceptual schema** to formulate queries while leaving the DBMS responsible for the execution strategy.

SQL is a relational language, meaning it manipulates tables through queries and generates tables as a result. It has evolved over time through standardized versions released by international standards organizations. Hereafter the main historical versions of SQL.

- SQL 1986: First official ANSI standard for SQL (*the foundation*)
- SQL 1989: Minor revisions; introduced integrity enhancement features.
- SQL 1992: the new standard with Major update: better joins, integrity constraints, data types.
- SQL 1999: Object-relational features, triggers, user-defined types, recursive queries.
- SQL 2003: XML support, window functions.
- SQL 2016: JSON support, row pattern matching, improvements to security and analytics.

Despite this evolution and the existence of different versions, SQL does not have the power of a real programming language. In fact, the input/output commands, assignments, conditional statements and loops structures are not allowed. Thus, for certain specific requirements, it is necessary to couple SQL with a more complete programming language, such as PL/SQL.

SQL offers various subsets of functions to satisfy different levels of users' requirements.

❖ **DDL**: Data Definition Language: Creating databases.
❖ **DML**:Data Manipulation Language: Data management (*inserting, updating and filtering data*).
❖ **DCL**: Data Control Language: to control access to data and users' permissions
❖ **TCL**: Transaction control language: for managing transactions.
❖ **SQL dynamic programming**: to construct and execute SQL statements at runtime.

In what follows, we expose the different functionalities provided by SQL.

3. Data definition: Managing relational tables

It provides the SQL commands used to define, create, modify and delete the structure of database objects, such as: tables, schemas, views, indexes, and more. Here after, we focus on the main important commands related to tables manipulation.

3.1. Table creation (CREATE)

The `CREATE TABLE` command is used to create SQL tables. Its syntax is given below.

`CREATE TABLE <table name> (<table element>+)`

A table element is either a column definition or a constraint definition:

```
<TABLE ELEMENT>::= <COLUMN DEFINITION> | <TABLE CONSTRAINT>
```

➢ **Column definition:** the syntax of the column definition is as follows.

```
<COLUMN DEFINITION>::= <COLUMN NAME> <DATA TYPE>
[<DEFAULT CLAUSE>] [<COLUMN CONSTRAINT>]
CREATE TABLE <table-name>
(column1 data_type,
column2 data_type, …);
```

Example 1: Creating `STUDENT` Table without constraints

```
CREATE TABLE STUDENT
student_ID int(10),
last name varchar(30),
first name varchar(30),
home address varchar(30) );
```

➢ **Column constraint:** Column constraint can be:

- Prohibition of null values for certain attributes by using the NOT NULL keyword.
- Uses the UNIQUE clause to ensure the uniqueness of attribute values (*no duplicate values in columns*).
- PRIMARY KEY: Primary key.
- DEFAULT value: used to specify the default value for a particular attribute.
- CHECK (*condition*) to verify that the attribute meets the condition when inserting tuples.

Example 2: Creating `CUSTOMER` Table with columns constraints

CUSTOMER (CUSTOMER-CODE, LAST-NAME, FIRST-NAME, AGE, CITY)

```
CREATE TABLE Customer (
CLIENT-CODE SMALLINT PRIMARY KEY,
LAST-NAME VARCHAR (20) NOT NULL,
FIRST-NAME VARCHAR (20) NOT NULL,
AGE SMALLINT CHECK (AGE >15),
CITY CHAR (35) NOT NULL);
```

- **Table constraint:** Table constraint is in the form:
  FOREIGN KEY (column…) REFERENCES TABLE [(column…)]

Example 3: Creating tables and linking them with foreign keys

CUSTOMER (CUSTOMER-ID, LAST-NAME, FIRST-NAME)
ORDER (ORDER-NUM, ORDER-DATE, #CLIENTID)

```
CREATE TABLE ORDER (
ORDER-NUM int NOT NULL,
ORDER-DATE DATE NOT NULL,
CUSTOMER-ID int,
PRIMARY KEY (ORDER-NUM),
FOREIGN KEY (CUSTOMER-ID) CUSTOMER REFERENCES (CUSTOMER-ID)
);
```

Example 4: Creating the PRODUCT table with various constraints

PRODUCT (ID, LABEL, PRICE, QUANTITY, STOR-ID, STOR-ADDR, VOLUME)

```
CREATE TABLE PRODUCT (
ID INT PRIMARY KEY,
LABEL VARCHAR (20) NOT NULL,
PRICE DECIMAL (10,2) CHECK (PRICE > 0),
QUANTITY INT CHECK (QUANTITY >=0),
STOR-ID INT,
STOR-ADDR VARCHAR (255),
VOLUME INT CHECK (VOLUME > 0),
FOREIGN KEY (STOR-ID) REFERENCES STORE(STOR-ID));
```

3.2. Database schema modification (ALTER, RENAME, DROP,)

SQL offers two commands for deleting and updating relational tables: `DROP TABLE` and `ALTER TABLE`.

3.2.1 Removing a table

SQL tables are deleted using the command `DROP TABLE` having the following syntax:

`DROP TABLE` <*`Table-name`*>;

Deleting a table removes both the structure of the table, all data it contains and the associated indexes.

Example

```
DROP TABLE Product; % Allows removing the table Product from
                     the database.
DROP TABLE ROOMS;   % Removes the table ROOMS from the database.
```

NB:

- A foreign key from another table must not reference the table to be deleted.
- Foreign keys that reference the table to be deleted can be deleted in Oracle by adding the keyword `CASCADE` at the end.

3.2.2 Renaming a table

The command `RENAME` allows renaming an existing relational table. Its syntax is the following.

`RENAME TABLE <Old-table name> TO < New-table-name>`;

Example: `RENAME` *`employees-department`* `TO` *`employees`*

However, the RENAME command is **not standard across all DBMS**. Some systems (like **Oracle** or **older MySQL versions**) use RENAME differently from ALTER TABLE. The most used command for updating a table is the command ALTER TABLE.

3.2.3 Table modification

To update the schema of a given table in the database, the command ***ALTER TABLE*** is deployed. The syntax of the command is as follows.

ALTER TABLE <*Table name*> Action;

Where action refers to the specific modification to be applied. The following actions can be conducted.

a. Add a column

ALTER TABLE <*Table name*> ADD *Att Type* |<NOT NULL>;

This command allows inserting the attribute "***Att***" having the type "***Type***" to the table "***Table name***".

Example

```
ALTER TABLE employees
ADD email VARCHAR (40);
```

b. Rename a column

ALTER TABLE <*Table name*> RENAME COLUMN *OLD-Att* TO *NEW-Att*;

Allows changing the name of the attribute "***OLD-Att***" to "***NEW-Att***".

Example

```
ALTER TABLE students RENAME COLUMN fullname TO student-name;
```

c. Drop a column

ALTER TABLE <*Table name*> DROP COLUMN *Att*;

This command removes the attribute "***Att***" from the table "***Table name***".

Example

```
ALTER TABLE students DROP COLUMN Middle-name;
```

d. Adding a constraint

To enhance a table specification by adding a new constraint, the following command is used.

ALTER TABLE <*Table name*> ADD CONSTRAINT *Constraint-name*;

Example

```
ALTER TABLE products
ADD CONSTRAINT price CHECK (price > 0);
```

The constraints-name may involve adding primary or foreign keys.

Adding a Primary key

```
ALTER TABLE employees
ADD CONSTRAINT pk_employee_id PRIMARY KEY (employee_id);
```

Adding a Foreign key

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer_id FOREIGN KEY (customer_id)
REFERENCES customers(customer_id);
```

e. Removing keys

Primary and foreign keys can be deleted by deploying the command ALTER TABLE with the action: `DROP PRIMARY KEY` and `DROP FOREIGN KEY`.

Examples

- **Drop Primary Key**

```
ALTER TABLE employees
DROP PRIMARY KEY;
```

- **Drop Foreign Key**

```
ALTER TABLE orders
DROP FOREIGN KEY fk-customer-id;
```

4. Data manipulation (INSERT, UPDATE, DELETE)

In SQL, data manipulation is mainly handled through a set of commands used to change the data inside the tables of the database. They vary from inserting data to removing the from the database. The following commands are used.

4.1 Inserting data in a table (INSERT)

To add new rows (tuples) in a specific table the command `INSERT` is used. Its syntax is as follows.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

If the list of column names is not specified, all attributes of the relation must be provided in the order of declaration.

If only a few columns are specified, the others are inserted with the value NULL.

Examples

- **With Columns specification**

```
INSERT INTO client (Client-ID, first-name, last-name, age)
VALUES ('100', 'MAHDOUS', 'Kamel', 20)
```

- **Without columns specification**

```
INSERT INTO client () VALUES ('100', 'MAHDOUS', 'Kamel')
```

- **Inserting multiple lines**

```
INSERT INTO client (Client-ID, first-name, last-name, age)
VALUES ('100', 'SAADAOUI', 'Rachda', 30),
       ('200', 'MAHDOUS', 'Safi', 40),
       ('300', 'NOUIDER', 'Saber', 55)
```

4.2 Updating data (UPDATE)

The `UPDATE` command allows modifying the values of attributes within the existing tuples.

The modification can affect all tuples in the table by providing values to be changed, or by constructing values via an expression. The syntax of the command is given below.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Where ***condition*** specifies the condition to be satisfied by the records to be modified.

Examples

- **Changing the address of the client « MAHDOUS Safi»**

```
UPDATE CLIENT
SET address= 'Cité des amendiers - Guelma'
WHERE Last-Name=' MAHDOUS ' AND First-name='Safi'
```

- **Add two points to students who have a mark under 8.**

```
UPDATE Student
SET mark=mark+2
WHERE mark<8
```

4.3 Deleting Data (DELETE)

The `DELETE` command allows removing existing data from a table. Its syntax is as follows.

```
DELETE FROM table_name WHERE condition
```

Where `table_name` specifies the name of the table from which records are to be deleted, and condition describes the condition that must be met for a record to be deleted. If no condition is specified, all records in the table will be deleted.

Examples

- **Removing all employees whose salary is less than 30,000 DA**

```
DELETE FROM employees
WHERE salary < 30000;
```

- **Removing all employees living in Guelma and whose salary is less than 30,000 DA.**

```
DELETE FROM employees
WHERE salary < 30000 and City ='Guelma';
```

5. Selection queries (SELECT, WHERE, GROUP BY, ORDER BY, HAVING)

The SQL language offers an efficient mechanism for data selection. In fact, data filtering is mostly done using the SELECT statement. It performs a simple and powerful tool for selecting data in the database that satisfy various users' requirements.
The general syntax of the `SELECT` statement is given bellow.

```
SELECT [DISTINCT] column1, column2, ...
FROM table_name
[WHERE condition]
[GROUP BY column]
[HAVING condition]
[ORDER BY column [ASC|DESC]]
[LIMIT number];
```

The different parameters of the command are explained below.

- `SELECT`: list the columns that users want to retrieve.

The symbol * means that all the columns are displayed.

- `DISTINCT`: is an optional parameter that removes duplicate rows.
- `FROM`: specify the table (*or tables*) users want to get data from.
- `WHERE`: is an optional filter that describe a condition on which rows are filtered.

Omitting the WHERE clause allows displaying the entire table (all the record)

- `GROUP BY`: an optional parameter for group rows having the same values in specified columns.
- `HAVING`: an optional filter groups (works like WHERE, but after grouping).
- `ORDER BY`: optional parameter for sorting the results by one or more columns.
- `LIMIT`: optional parameter that restricts the number of rows returned.

Examples

The illustrative queries are based on the following schema of tables **CUSTOMERS** and **ORDER**.

CUSTOMER (Customer-Code, First-name, Last-name, Age, City, email)

ORDER (Order-ID, Customer-code, Total-amount)

- **Displaying information of all customers.**

```
SELECT Customer-Code,First_name, Last_name, Age, City, email
FROM customer;
```

- **Find customers living in a specific city (***Guelma, for example***)**

```
SELECT first_name, last_name
FROM customer
WHERE city = 'Guelma;
```

- **Select customers with the last-name "*IDRISSI*" with age greater than 60 years.**

```
SELECT *      % "*" to display all the attributes of the table;
FROM Customer
WHERE age > 60 and Last-name = 'IDRISSI';
```

- **Select customers with age greater than 40 years sorted by last-name.**

```
SELECT *      % "*" to display all the attributes of the table;
FROM Customer
WHERE age > 60
SORTED BY LAST-NAME % Ascending is the default value
```

- **Get the top 5 customers by total spending**

```
SELECT c.first_name, c.last_name, SUM(o.total_amount) AS
total_spent
FROM customer c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name
ORDER BY total_spent DESC
LIMIT 5;
```

6. SQL Aggregation Queries (AVG, COUNT, MAX, MIN, SUM)

To facilitate decision making by conducting data analytics, the SQL language offers a set of integrated functions. The general syntax of the aggregate function is the following.

```
SELECT AGGREGATE_FUNCTION(column_name)
FROM table_name
[WHERE condition];
```

The table bellow summarizes the aggregate functions.

Aggregate Function	Description
AVG()	Calculates the average (mean) value of an attribute
COUNT()	Counts the number of rows
MAX()	Finds the maximum (highest) value of an attribute
MIN()	Finds the minimum (lowest) value of an attribute
SUM()	Adds up (sums) all the values of an attribute

Examples

- **Compute the mean of all invoice totals.**

```
SELECT AVG (total_amount) AS average_invoice
FROM invoices;
```

- **Compute the number of customers.**

  The command `COUNT(*)` counts all rows of the considered table.

```
SELECT COUNT(customer_id) AS number_of_customers
FROM customers;
```

- **Find the biggest invoice total.**

```
SELECT MAX(total_amount) AS highest_invoice
FROM invoices;
```

- **Calculate total sales made**

```
SELECT SUM(total_amount) AS total_sales
FROM invoices;
```

- **Find the total sales per city (Combining aggregates with `GROUP BY`)**

```
SELECT city, SUM (total_amount) AS city_sales
FROM customers
JOIN invoices ON customers.customer_id = invoices.customer_id
GROUP BY city;
```

7. Join and sub-queries

As explained in the previous chapter, the relational algebra manipulates the join operations to link different tables in a database. In SQL, **joins** and **sub-queries** are two fundamental techniques for combining and filtering data from multiple tables. Here after we give the different structures of the **join** command and we illustrate their usage on various examples.

7.1 Natural join

The natural join operator automatically joins tables using all columns with the same names and compatible data types. This operator is implemented in SQL without explicitly specify the join condition. Its syntax is as follows.

```
SELECT * FROM table1 NATURAL JOIN table2;
```

Example

Let the two tables Student and module related to the s**cholarly (academic)** context.

Student

ID	First-Name	Last-Name	Section-ID
100	Mohamed	MEZDOURI	ING2
200	Said	NESSAIRI	LMD3
300	Nassima	SOUIDI	M1

Section

Section-ID	Description
ING2	Engineer 2nd level
LMD3	LMD license 3rd level
M1	Master 1st level

The following query ensures the join between the two tables

```
SELECT *
FROM students
NATURAL JOIN section;
```

This query links the two tables basing on the common attribute (*Section-ID*) and its result is given below.

ID	First-Name	Last-Name	Section-ID	Description
100	Mohamed	MEZDOURI	ING2	Engineer 2nd level
200	Said	NESSAIRI	LMD3	LMD license 3rd level
300	Nassima	SOUIDI	M1	Master 1st level

7.2 Inner join

An **`INNER JOIN`** in SQL is a type of join that returns only the rows where there is a match in both tables based on a given condition. The syntax of the command is as follows.

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Or by using the clause WHERE

```
SELECT *
FROM table1, table2
WHERE table1.colonne_table1 = table2.colonne_table2;
```

Example

Let us consider the two tables: Student and enrollments with their respective tuples.

Student

ID	First-Name	Last-Name	Section-ID
100	Mohamed	MEZDOURI	ING2
200	Said	NESSAIRI	LMD3
300	Nassima	SOUIDI	M1

Enrollments

Student-ID	Course-Code
200	DB
300	ADS
400	OS

The following query allows joining the two tables basing on the student ID.

```
SELECT student.first-name, last-name, enrollments.course-code
FROM students
INNER JOIN enrollments
ON student.ID = enrollments.student-ID;
```

The result of the query.

First-Name	Last-Name	Course-Code
Said	NESSAIRI	DB
Nassima	SOUIDI	ADS

The same query can be expressed with the clause WHERE.

```
SELECT student.first-name, last-name, enrollments.course-code
FROM students, enrollments
WHERE student.ID = enrollments.Student-ID;
```

7.3 Left join

In the left outer join, noted `LEFT-JOIN`, we keep all the tuples from the first (*left*) table. If no match exists, columns from the right table are `NULL`.

Example

```
SELECT *
FROM Student
LEFT JOIN Enrollments
ON Student.ID = Enrollments.Student-ID;
```

The previous query gives the following table as a result.

ID	First-Name	Last-Name	Section-ID	Student-ID	Course-Code
100	Mohamed	MEZDOURI	ING2	NULL	NULL
200	Said	NESSAIRI	LMD3	200	DB
300	Nassima	SOUIDI	M1	300	ADS

7.4 Right Join

In the right outer join noted `RIGHT-JOIN`, we keep all the tuples from the second table (*Right*). If no match exists, columns from the right table are NULL.

Example

```
SELECT *
FROM Student
RIGHT JOIN Enrollments
ON Student.ID = Enrollments.Student-ID;
```

The result of this query includes all rows from **`Enrollments`**, and matching rows from `Student`. If no match exists, columns from `Student` are set to the `NULL` value.

ID	First-Name	Last-Name	Section-ID	Student-ID	Course-Code
200	Said	NESSAIRI	LMD3	200	DB
300	Nassima	SOUIDI	M1	300	ADS
NULL	NULL	NULL	NULL	400	OS

8. Conclusion

In this chapter, an important step in learning how to communicate with relational databases using **SQL** is exposed. We have illustrated that SQL is much more than just a programming language, but it's a powerful tool that helps users create, organize, and retrieve data efficiently.

We started by learning how to define tables and set rules to keep the data consistent using **DDL** commands. Then, we looked at how to update the database by specifying queries using **add**, **change**, and **remove** data offered by the **DML** commands. Also, we have discussed how to explore the database for retrieving information by formulating adequate queries using the **SELECT** statement and how to narrow down results with filters, sort them, or group them to get summaries like totals and averages. Finally, a set of aggregated functions are explained and highlighted to support managers decision-making activities.

We also explored how data from different tables can be connected using **joins**, which are very useful in real-world databases where information is spread across several tables.

Series of exercises No. 4

Exercise 1

Consider the relational database of an airline's company containing the following tables: **Plane**, **Pilot**, and **Flight**. The structure of each table is described below.

Plane Table (Plan-N: Integer plane number (primary key),
P-Name: Text plane name (12),
Capacity: Integer plane capacity (3),
Location: Text plane city (10)
)

Pilot Table (Pilot-N: Integer pilot number,
Name: Text pilot name (25),
Address: Text pilot address (40)
)

Flight Table (FN: Text flight number (6),
Pilot-N: Integer pilot number,
Plan-N: Integer plan number,
DC: Text departure city (10),
AC: Text arrival city (10),
DT: Integer departure time,
AT: Integer arrival time)
)

1. Formulate the SQL command to create the plane table reinforced with possible constraints.

2. Insert the following data into the plane table:
(100, AIRBUS, 300, ALGER), (101, B737, 250, ANNABA), (102, B737, 150, ORAN)

3. Modify the capacity of the plane number 101 with the new capacity 220.

4. Delete planes having a capacity less than 200.

Exercise 2

Consider the relational schema for managing customer's loans in a bank.

Customer (Customer -Num, Last-Num, First-Num, City-Customer)

Account (Account-Num, Customer -Num, Branch-Num, Balance)

Branch (Branch-Num, Branch-Name, Branch-City)

Loan (Employee-Num, Customer-Num, Branch-Num, Amount)

1. Create all tables with PK (primary Key) and FK (foreign Key) integrity constraints.

2. Give the SQL command for inserting one record per table.

3. Formulate the commands to satisfy the users' requirements for inserting the following data into the database:
Branch: (1, 'BRANCH1', 'Guelma'); **Customer**: (1, 'Selmaoui', 'Mahmoud', 'Annaba')

Account:(1, 2, 3, 25000); **Loan** (1, 1, 1, 60000)

4. Add a strictly positive constraint (>) for the attribute Amount.

5. Change the Null value of the amounts to 0.

6. Change the Client cities to lowercase.

7. Increase the balance of all clients by "0.5%"

8. Delete the attribute "City-Customer" from the Customer table and add the email attribute.

Exercise 3

Let us consider the relational model of the school management database of the exercise **N° 1 in serie 3**.

Student (Student ID, Name, Address)
Course (Course ID, Day, Time)
Assign (Course ID, Room ID)
Evaluate (Course ID, Student ID, Mark)

Express the following queries in SQL.

Q1: List of students with their addresses.
Q2: What is the teaching time for the DB course?
Q3: List the identifiers of students having obtained more than the average in the DB course.

Exercise 4

Consider the database schema for the company's employee management of exercise 2 in serie 3.

EMPLOYEE (E-Num, E-Name, Profession, Hire-Date, Salary, Commission, #Dep-Num)
DEPARTMENT (#Dep-Num, Dep-Name, Director, City-D)

Formulate the SQL queries for expressing the following requirements:

Q1: List of engineers (Number, Name) hired before 01/01/2020, along with their commission.

Q2: Give all tuples of EMPLOYEE.

Q3: Give the hire dates of the technicians.

Q4: The name and profession of employee number 10.

Q5: Give the employees' professions (after eliminating redundant tuples).

Q6: Give the names of the employees and their departments.

Q7: Give the numbers of the employees working at ANNABA.

Q8: Give the names of the managers of departments 1 and 3.

Q9: Give the names of the employees working in a department with at least one engineer.

Exercise 5

Consider the relational schema for department management consisting of the two relations.

- EMPLOYEES (Emp_Num, Emp_Name, Position, Salary)
- DEPARTMENTS (Dep_Num, City)

Express the following queries in SQL.

1. What is the name, position, and salary of the employee(s) with the highest salary?
2. What is the number of different positions held in department 30?
3. What is the average salary per department?
4. What is the average annual salary (12 months) of all employees for each department except those with the position of 'MANAGER' or 'PRESIDENT'?
5. What are the numbers of departments with more than 10 employees?
6. What are the different positions held in all departments and the average salary per position?
7. What are the different functions performed in all departments whose average salary per function is greater than 10,000?
8. List the employees in Department 10 in descending order of salary?
9. List the employees sorted alphabetically by function, and within each function sorted by descending salary.

Annex: Solutions to the series of exercises

Solution to exercises in series No. 1

Exercise 1

The file management systems (FMS) present different limitations.

a. The main drawbacks of FMS are the following.

- **Data inconsistencies:** FMSs often do not enforce data validation rules or constraints, leading to potential data inconsistencies.
- **Information redundancy issues:** some data can occur more than once in different files.
- **Scalability issues:** FMSs are limited when managing a large mass of data containing various links between them which leads to an increasing access time to an unplanned query.
- **Storage mode application dependency:** the applications are dependent on the data storage mode and the user must know the physical structure of the records, the access mode, and the location of the used files.
- **Data heterogeneity and weak integrity:** data is stored in different formats. For example, the phone number has character type in the customer file and an integer type in the accounting file. Furthermore, difficulty of imposing constraints on data. For example, the product price must never be less than 0.
- **Storage mode user need dependcy:** for each new user need a specific program is required.
- **Lack of security:** as files are acceeded by multiple users and several programmers, data security and unauthorized access are not guaranteed.
- **Concurrency Problem:** if multiple users access the same files at the same time, then concurrency problems arise, leading to data inconsistency.

b. How can FMS limitations be overcome ?

The limitations of File Management Systems (FMS) were addressed through the development of **Database Management Systems (DBMS)** which introduced centralized, structured, and secured ways to manage data.

c. Data redundancy may induce the following problems

- Loss of memory space due to the multiple storage of the same information.
- Increased access time for searching information due to its replication that leads to a huge size of manipulated files.
- Multiple updated for the same information stored in different files.
- Data inconsistency and anomalies. If the data is replicated, we don't know what is the right one.

Example: Let's consider the **product price** information which is manipulated by the following files and applications:

- The main products file of the inventory management application,
- The sale file used by the sales and marketing application,
- The customer account managed by the financial field,

Consider now that we want to modify the **product price** which occurs in three different files. In this case it's imperative to modify the information in all the three files at the same time, in order to avoid update anomalies, otherwises the right product price can't be known.

Exercise 2

Database fundation is based on the principle of separating the user vision into different abstract levels.

a. The purpose of separating the view of databases into several abstraction levels

Separating the database framework into abstraction levels serves to enhance usability and flexibility by offering modular systems that can meet diverse user requirements, adapt to changes, and maintain efficiency. The main benefits of this separation are listed above.

- **Simplifies interaction for different users:** end users can work with a simplified and user-friendly representation of data (*external level*), while developers and administrators can manage the database's underlying structure and physical storage without affecting user views (*internal level*).
- **Ensuring data independency:** the separation allows changes to one level of the database without affecting the others. Hence, changes to the conceptual schema (*e.g., adding a new attribute*) do not affect external schemas, and changes to the internal schema (*e.g., reorganizing storage*) do not affect the conceptual or external schemas.
- **Enhancing data security:** by showing only relevant portions of the database to users. Thus, users at the external level see only the data they are authorized to access.
- **Improves Flexibility:** adaptation to changing requirements and applications are easier and external schemas can be tailored for specific user groups or applications without modifying the underlying conceptual schema.

b. The conceptual level and how it is approached

The conceptual level involves creating a conceptual schema or a model of the future database according to a particular data model. The conceived model will serve as blueprint of the database's structure. Thus, it integrates all the manipulated data, their constraints and their relationships.

The conceptual level is crucial for ensuring a clear, consistent, and robust database design that meets organizational goals and supports data independence. Hence it is approached according to the following four steps.

i. Understanding and gathering the data needs of the organization.
ii. Defining constraints (*primary keys foreign keys, uniqueness, nullability, ...*) to ensure data integrity and consistency.
iii. Data modeling by creating a logical representation of the manipulated data.
iv. Validation of the conceived model by checking that it meets requirements and aligns with real-world data usage.

c. Illustration of the external level with a real word scenario

This external level ensures that the system meets each user group interacting with the database. As a real-world example consider the database for managing data related to **students**, **courses**, **faculty**, and **departments**. Different stakeholders (*e.g., students, professors, administrators*) need access to this data, but their needs differ significantly.

- **Student View**: Allow students to view and manage their academic information (*e.g., StudentID, Name, Enrolled Courses, ...etc.*), while hiding other information like: *Faculty salaries, and administrative reports*.
- **Professor View**: Enable professors to access and manage information related to their teaching (*e.g., CourseID, Course Name, Enrolled Students ...etc.*), but students' *payment details, or personal faculty details* are hidden.

- **Administrator View**: Allow administrators to manage the entire database for operational needs. Hence, all data including *student payments, faculty salaries, course schedules, and department budgets are accessible*.

Exercise 3 : Draw a comparison between network and hierarchical data models, basing on the following features: structure, relationship, data redundancy, data access, query complexity.

Comparison features	Hierarchical data model	Network data model
Structure	Tree-like structure. Each parent node can have multiple child nodes, but each child has only one parent.	Graph structure. Nodes (*records*) can have multiple parent and child relationships (*many-to-many*).
Relationships	**Rigid** Limited to simple hierarchical relationships. One-to-many (1:N)	**Flexible** Supports more complex and interconnected relationships. Many-to-many (M:N).
Data Redundancy	Higher redundancy due to strict hierarchy.	Less redundancy because of shared relationships.
Data access	Navigates through a single predefined path (*top-down*).	Navigates through multiple paths using pointers.
Query Complexity	Simpler queries for hierarchical relationships.	More complex queries due to interconnected data.

Exercise 4 : DBMS are required to ensure the interaction betwwen users and the database.

a. A brief description of the different languages performed by a DBMS

The DBMSs offer different languages enabling users and administrators to perform essential tasks related to data management. The following languages are performed by these systems.

- Data Definition Language (**DDL**): its purpose consists to define and manage the structure of the database.
- Data Manipulation Language (**DML**)/ is used to manipulate and manage data within the database.
- Data Query Language (**DQL**): focuses solely on querying and retrieving data from the database.
- Data Control Language (**DCL**): Controls access to the database and manages permissions.

b. The consequences of DBMS crash

The consequences of a DBMS crash can range from minor inconveniences to major disruptions. The most important one is data loss. Here after, we discuss some problems generated after a DBMS crash.

- **Loss of data:** recent changes or transactions not yet committed to permanent storage, may be lost during a crash.
- **Data corruption:** during write operations or while modifying the database structure (*e.g., adding a table or index*) can leave the database in an inconsistent state, such as incomplete or corrupted records, damaged indexes leading to errors in queries
- **Inconsistent database state:** some parts of a transaction might succeed while others fail, leaving the database in an invalid state. For example, in a banking transaction, money may be debited from one account but not credited to another.
- **Security Vulnerabilities:** leading to compromises security features. In this case, unauthorized users might exploit the system during recovery or maintenance.

c. Different solutions to avoid problems induced by a DBMS crash

To avoid the problems induced by a DBMS crash, various preventive measures and solutions can be implemented at the database, application, and infrastructure levels. These solutions aim to enhance data integrity**,** availability**,** and reliability. Here are the key possible strategies:

- **Database replication: c**reate copies of the database to ensure its availability and to reduce downtime.
- **Regular backups:** ensure that the database can be restored to a known good state after a particular crash.
- **Database partitioning (sharing): t**his action can be achieved by splitting the database into smaller, manageable pieces (*partitions*) to improve performance and reduce the impact of crashes.
- **Monitoring and alerts: p**roactively detect problems and prevent crashes or mitigate their impact before they happen. For example, send alerts to the database manager for events like low disk space, high CPU usage, or application bugs

Exercise 5 : Although the E/A data model provides a framework for the database, it is not directly implementable in a DBMS.

a. The concepts manipulated by the E/A model are:

- **Attributes:** are the properties or characteristics of an entity
- **Entities:** refers to objects (*person, place, things*) in the real world that are distinguishable from other objects and has significance for the system being modeled.
- **Relationship:** represent associations between two or more entities. They describe how entities are linked to each other.
- **Keys:** particular attributes used to uniquely identify an entity or a relationship. The most important key in the E/A model is the **primary key**.
- **Cardinality:** defines the number of instances of one entity that can or must be associated with each instance of another entity in a relationship.

b. Types of relations handled by the E/A data model

The E/A data model allows expressing various types of relations.

- **One-to-One (1:1)**: an instance of entity A is related to at most one instance of entity B, and vice versa. For example, a student belongs to only one group.
- **One-to-Many (1:M)**: an instance of entity A can relate to multiple instances of entity B, but an instance of entity B is related to only one instance of entity A. For example, a product occurs in different customers' orders.
- **Many-to-Many (M: N)**: Instances of entity A can relate to multiple instances of entity B, and vice versa. For example, a product can be stored in multiple stores, and the store may contain multiple products.

c. Exploitation of the E/A data model for managing data by DBMS.

By clearly defining entities, attributes, relationships, and constraints, the E/A model ensures that data is structured logically and consistently. Hence, it provides a powerful conceptual framework for organizing, managing, and maintaining data in relational databases. However, as it's an abstract and conceptual model, it can't be directly exploited by commercial DBMS. In fact, the model lacks of implementation details, and it does not address physical storage or performance optimizations. Consequently, it requires translation into a physical data model (*e.g., relational schema*) for its concrete implementation.

Solution to exercises in series No. 2

Exercise 1: Let us consider the entity/association model relating to the management of an educational establishment.

1. Transformation of the E/A model to the relational model.

First, the entities of the model are transformed to relations. Then, the set of associations between entities are classified into 1-N and N-N types. After that, the transformation rules are directly applied.

Entities and associations of type parent-child (associations 1-N)

- Student (Student-code, Last Name, First Name, Date of birth, Class-ID);
- Class (Class-ID, Students-number, Classroom-code);
- Classroom (Classroom-code, capacity);
- Exam-Type (Exam-code, exam-designation);
- Professor (Prof-Code, Prof-name, grade, Category-code);
- Category (Category-code, category name);
- Module (Module-ID, module-name, coefficient).

Associations of type not parent-child (associations N-N: many to many)

- Evaluate (Student-code, Exam-code, Prof-code, Module-ID, Mark);
- Teaching (Prof-code, Module-ID, Class-ID, Hours).

2. A possible extension of the relation Teaching (Prof-code, Module-ID, Class-ID, Hours).

Extending the relation defines the set of tuples, by simply providing occurrences for each attribute, but in tabular form.

Prof-code	**Module-ID**	**Class-ID**	**Hours**
100	DB	ING2	4
200	OS	ING2	6
300	GT	L2LMD	3
100	DS	L1LMD	1.5

3. The key of the relation **Evaluate** (Student-code, Exam-code, Prof-code, Module-ID, **Mark**).

The key of the relation **Evaluate** is composed of four attributes: **Student-code, Exam-code, Prof-code,** and **module-ID.** Thus, it's an articulated or composed key.

Explanation: This key is resulting from the fact that the **Evaluate** association which is of type N-N, links 4 entities. Therefore, its transformation to the relational model will give a new relation having as key the composition of the identifiers of the entities participating in the association.

Exercise 2

1. Inserting the missing values in boxes of the table.

A	**B**	**C**	**D**	**E**	**F**
w	*1*	***i***	*110*	***m***	*54*
x	*2*	*j*	*100*	*n*	*52*
w	*1*	*i*	*110*	*m*	***54***
x	*2*	***j***	*100*	***n***	*52*

2. Let the relation ***Film (Title, Year, Length, filmType, StudioName, starName)*** with its extension illustrated by the set of tuples of the following table.

title	year	length	filmType	studioName	starName
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Esteves
Wayne's World	1992	95	color	Paramount	Dana Carvey
Wayne's World	1992	95	color	Paramount	Mike Meyers

a. Identification of FD types (trivial, canonical, elementary and direct).

Type of FD	FD	Counter examples
Trivial (*an attribute determines itself or a part of itself*)	title, year length → year	title ↛ starName
Canonical (*the target of the DF must be a single attribute*)	title → year	title ↛ year, length, filmType
Elementary (*The target of the DF does not depend on any part of the left part of the DF*)	title, length → year	title, length → Year title → year (year depends on a part of title, length).
Direct *(Elementary and not transitivite).*	title → year	title → year and year → studioName (by transitivity)

b. Identification of elementary FDs.

A FD: X → Y is elementary if Y does not depend on any part of X. So, we will identify all the FDs, where the left part of the FD completely determines the target part of the FD.

The elementary FDs in the table ***Film*** are the following:

Title → year, length, filmType, studioName.

As a counter example: the FDs title, year ↛ length, filmType, studioName are satisfied between attributes of the relation ***Film***, but are not elementary because the target attributes (length, *for example*) depend on a part of the left part of the FD (*the title*).

Title → Year, length, filmType, StudioName.

Canonical form of the elementary DF:

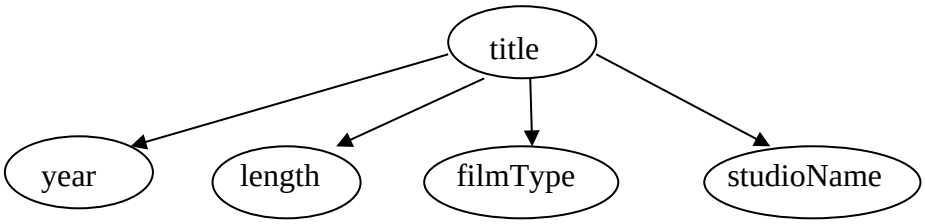
Title → year.

Title → length

Title → filmType.

Title → StudioName.

c. Graph of FD



Exercise 3

Consider the relation ***R (A, B, C)*** with the set of FD: {A → B; B → C}.

1. The primary key of R.
   As *A → B, C* (by transitivity), so the key of *R* is *A*.
2. The normal form of the relation *R.*
   The relation *R* is in 2NF. It doesn't satisfy the 3NF because the attribute C is a non-key attribute which depends on another non-key attribute B. Thus, C depends on A by transitivity).
3. Is the following extension of the relation R' an extension of R?

R'	A	B	C
	A1	B1	C1
	A2	B1	C2
	A3	B2	C1
	A4	B3	C3

The relation R' is not an extension of R because tuples of the relation R does not satisfy the FD: B → C. For instance, for the same value B1 of B, we find two different values of C (C1 and C2).

4. An extension of the relation ***R'*** that conforms to ***R***.

R''

A	B	C
A1	B1	C1
A2	B1	**C1**
A3	B2	C1
A4	B3	C3

5. Decomposition of ***R*** in in the 3NF without loss of information.
   R1(A, B) and R2 (B, C).

Exercise 4

ORDER (Order-ID, Order-date, Customer-ID, Cust-address, Product-ID, Price, Quantity)

Order-ID	Order-date	Customer-code	Cust-address	Product-ID	Price	Quantity
C217	26/03/2025	CUS423	Berrahal-Annaba	P983	200	575
C902	14/10/2025	CUS611	Sidi-rached-Setif	P120	650	575
C465	10/10/2024	CUS423	Berrahal-annaba	P482	650	88
C510	14/10/2025	CUS064	Nouvelle ville-Annaba	P983	200	204
C510	14/10/2025	CUS064	Nouvelle ville-Annaba	P005	4700	196

1. The key of the relation ***Order.***

Before finding the key, we must first determine the set of FDs existing between attributes of the table. The statement doesn't mention any business rules, but we can assume the following FDs:

a) Each order is placed by a single customer with a given address and on a given date.
   So, we have the FD:
   Order-ID → Order-date, Customer-code, Cust-Address.
b) A customer has a single address. So, we have the FD:
   Customer-code → Cust-Address
c) Each product has a single price for the various orders.
   So, we have the FD:
   Product-ID → Product-price
d) For a given product, a specific quantity is specified in a given order.
   So, we have the FD:
   Order-ID, Product-ID → Quantity.

Basing on the previous FDs set, we can conclude that the two attributes: Order-ID, Product-ID determine all the other attributes. Thus, they constitute the key of the relation **Order**.
Order (**Order-ID, Product-ID,** Order-date, Customer-ID, Cust-address, Price, Quantity)

2. The normal form of the ***Order*** relation.
   According to the FDs (a) and (c), where the attributes partially depend on the key "***Order-ID, Product-ID*** " of the relation. Hence, the relation ***Order*** is in 1NF but not in 2NF.
3. Normalization of the relation ***Order*** into the 3NF.

   Conversion process to 3NF:
   When the relationship is in 1NF, it must be converted to 2NF before moving to 3NF.
   a. The process of converting to 2NF
   Keep the dependent attributes of the entire key in the initial relation, we obtain a new relation:

   Order-Product **(***Order_ID, Product-ID,* Quantity**)**

   b. Group the attributes depending on a part of the primary key into new relations, and make these parts of the key the primary key of the new relations:
      Order (*Order_ID*, *Order-date*, Customer-ID, Cust-address)
      Product (*Product-ID*, *Price*)
   c. Transformation process to 3NF process:
   The relations and ***Order-Product*** and ***Product*** are in 3NF. However, in the ***Order*** relation, the attribute " Cust-address " does not directly depend on the " *Order_ID* " key. The customer address depends on the " *Customer-ID*" attribute.

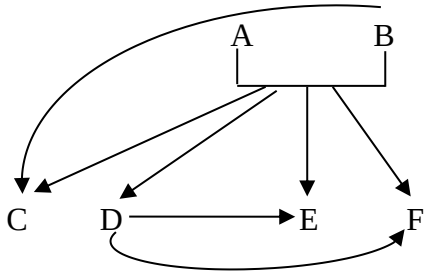
   To normalize the relation ***Order***, we group the attributes: *Customer-ID* and *Cust-Address* into a new relation, where the "*Customer-ID* " attribute becomes the primary key and remains duplicated in the ***Order*** relationship.
   The database schema satisfying the 3NF is the following.

   Product (*Product-ID, Price*)
   Order-Product (Order-ID, Product-ID, Quantity).
   Order (*Order_ID*, *Order-date*, Customer-ID)
   Customer (*Customer-ID, Cust-address)*

Exercise 5

Let ***R1 (A, B, C, D, E, F)*** be a relation having the following set of functional dependency:
{A, B → C; A, B → D; A, B → E; A, B → F; B → C; D → E; D → F}.

1. The functional dependency graph associated to ***R1***.



2. The set of minimal functional dependencies is the following.

Min= {AB → D ; B → C ; D → E, D → F}.

The key of this relation is the pair (**A, B**), because with the two attributes A and B we can determine all the other attributes of the relation R1.

3. The normal form of the relation ***R1***

The relation ***R1*** is in first normal form (**1NF**) but not in second normal form because the attribute C depends on part of the key: (A, B), according to the FD: B → C). (*A non-key attribute depends on a part of the key*).

4. We decompose the relation ***R1*** in two separate relations ***R11*** et ***R12,*** *with:*

R11 (A, B, D, E, F) *and* ***R12 (B, C)***.

The normal forms of ***R11*** and ***R12***:

- The relation ***R11*** is in 2NF, but not in 3 NF, because it remains some transitive FDs in the following residual set of FDs.

  {A, B → D; A, B → E; A, B → F; D → E; D → F}.

- The relation ***R12*** is in BCNF.

5. Decomposition of R11 without information loss.

R11 can be decomposed into ***R111*** (A, B, D) and ***R112*** (D, E, F) without information loss.

Solution to exercises in series No. 3

Exercise 1

1. The following projections produce as result the tables below.

   Q1 = $\pi_{\text{Course ID}}$ (Course)

Course ID
Archi
ASD
DB

Q2 = $\boldsymbol{\pi}_{\text{Student ID, Address}}$ (Student)

Student ID	Address
100	Guelma
200	Skikda
300	Annaba

2. The result of the query Q3 of the restriction $\sigma_{\text{Course ID = 'ASD'}}$ (Evaluate) is the following.

Course ID	Student ID	Mark
ASD	100	09
ASD	200	11
ASD	300	06

3. Expression of the queries Q4, Q5 and Q6 in the relational algebra.
   - Q4: List of students with their addresses.

     Q4 = $\boldsymbol{\pi}_{\text{Name, Address}}$ (Student)
   - Q5: the teaching time for the DB course. The query can be expressed by two sub-queries or in a single query.

     Q51 : $\boldsymbol{\sigma}_{\text{Cours ID = 'DB'}}$ (Course)

     Q52 : $\boldsymbol{\pi}_{\text{course ID,, Time}}$ (Q51)

     Or in one pass: Q5 : $\boldsymbol{\pi}_{\text{course ID,, Time}}$ ($\boldsymbol{\sigma}_{\text{Course ID = 'DB'}}$ (Course))
   - Q6: Students' identifiers having obtained more than the average in the DB course.

     Q6: $\boldsymbol{\pi}_{\text{Student ID}}$ ($\boldsymbol{\sigma}$ ($_{\text{Course ID = 'DB' and Mark >=10)}}$ (Evaluate))

Exercise 2

Consider the database schema for a company's employee management.

EMPLOYEE (E-Num, E-Name, Profession, Hire-Date, Salary, Commission, #Dep-Num)

DEPARTMENT (#Dep-Num, Dep-Name, Director, City-D)

Formulation of the requested queries:

- Q1: List of engineers (Number, Name) hired before 01/01/2020, along with their commission.

Q1: $\boldsymbol{\Pi}_{\texttt{E-Num,E-Name,Commission}}$ ($\boldsymbol{\sigma}_{\texttt{Profession='engineers'}\wedge\texttt{Hire-date <= '01/01/2020'}}$ (**EMPLOYEE**)).

- Q2: Query for listing all the tuples of the table EMPLOYEE.

Q2: *EMPLOYEE*

This expression retrieves **all tuples** (*i.e., all rows*) and **all attributes** (*i.e., all columns*) from the *EMPLOYEE* relation.

- Q3: Give the hire dates of the technicians.

$$\textbf{Q3 : } \Pi_{\text{E-Num, E-Name, Hire-date}} (\sigma_{\text{Profession}} = \text{'technicians'} (\textbf{EMPLOYEE}))$$

- Q4: The name and profession of employee number 10.

$$\Pi_{\text{E-Name, Profession}} (\sigma_{\text{E-Num}} = 10 (\textbf{EMPLOYEE}))$$

- Q5: Give the employees' professions.

$$\Pi_{\text{Profession}} (\textbf{EMPLOYEE})$$

Exercise 3

1. The tuples produced by the natural join between R and S are depicted in the following table.

   R ⋈ S

A	B	C
a	b	c
a	b	d
c	b	c
c	b	d
d	e	a

2. The equi-join: $\sigma_{A=C} (\rho_{B'/B} (R) \times S)$ (Equi-join; ρ denotes the renaming operator and X the Cartesian product)

$\sigma_{A=C} (\rho_{B'/B} (R) \times S)$

A	B	B'	C
a	b	e	a
a	f	e	a
c	b	b	c
d	e	b	d

3. $R \triangleright S = \pi_R (R \bowtie S)$ (Semi-join)

A	B
a	b
c	b
d	e

4. As shown in the two following tables, the equalities: $\pi_{A,B} (R \bowtie S) = R$ and $\pi_{B,C} (R \bowtie S) = S$ are not satisfied.

$\pi_{A,B}(R \bowtie S)=R$

A	B
a	b
c	b
d	e

$\pi_{B,C}(R \bowtie S)=S$

B	C
b	c
b	d
e	a

Exercise 4

The database schema of **Exercise 2**.

EMPLOYEE (E-Num, E-Name, Profession, Hire-Date, Salary, Commission, Dep-Num)

DEPARTMENT (Dep-Num, Dep-Name, Director, City-D)

Queries in relational algebra for meeting the needed requirements:

- Q1: Names of the employees and their departments.

$$\Pi_{\text{E-Name, Dep-Name}}(\text{EMPLOYEE} \bowtie \text{DEPARTMENT})$$

- Q2: The employee numbers working at ANNABA.

$$\Pi_{\text{E-Num}}(\text{EMPLOYEE} \bowtie \sigma_{\text{VILLE='ANNABA'}}(\text{DEPARTMENT}))$$

- Q3: Names of the Directors of Departments 1 and 3.
  NB: By careful: Director is not a profession !

$$\pi_{\text{E-Name}}(\sigma_{\text{Dep-Num=1} \vee \text{Dep-Num=3}}(\text{DEPARTMENT}) \bowtie_{\textit{DIRECTOR=E-NUM}} (\text{EMPLOYEE}))$$

- Q4: Names of the employees working in a department with at least one engineer.

$$R1 := \pi_{\text{Dep-Num}}(\sigma_{\text{Profession='INGINEER'}}(\text{EMPLOYEE}))$$

$$R2 := \pi_{\text{E-Name}}(\text{EMPLOYEE} \bowtie R1)$$

Exercise 5

STU

RESIDENT

ID	Residence	Room
200	1000 beds	A20
500	2000 beds	B20
400	1000 beds	C15

Expression of queries in the relational algebra.

1. Q1: *The numbers of the residents who are not students*:
   First of all, the two relations: Student and Resident must be transformed in order to have the same schema. This action is satisfied by applying the projection operator on the common attribute "ID". After that, the difference operator is applied.

$$\textbf{Q11} = \Pi_{\text{ID}}(\textbf{STUDENT}), \text{ and } \textbf{Q12}=\Pi_{\text{ID}}(\textbf{RESIDENT})$$

   After that: $\textbf{Q1= Q12 - Q11}$ (*The difference operator*)

2. Q2: "*The list of students (numbers, name), with the residence and the room number of each one*?" Perform a natural join, then a projection.

$$Q21: \text{STUDENT} \bowtie \text{RESIDENT}$$

Then operate a projection

$$Q2: \Pi_{\text{ID, name, Residence, Room}} (Q21)$$

3. Q3: "*The names of the students who reside in the " Springer " residence with their room numbers?*"

$$\Pi_{\text{Name, Room}} (\sigma_{\text{Residence= "Springer"}} (\text{STUDENT}) \bowtie (\text{RESIDENT}))$$

Solution to exercises in series No. 4

Exercise 1

Plane Table (Plan-N: Integer plane number (primary key),
P-Name: Text plane name (12),
Capacity: Integer plane capacity (3),
Producer: Text plane Producer (10)
)
Pilot Table (Pilot-N: Integer pilot number,
Name: Text pilot name (25),
Address: Text pilot address (40)
)
Flight Table (FN: Text flight number (6),
Pilot-N: Integer pilot number,
Plan-N: Integer plan number,
DC: Text departure city (10),
AC: Text arrival city (10),
DT: Integer departure time,
AT: Integer arrival time)
)

1. Creating the table Plane

```
CREATE TABLE Plan (
   P-Num    INTEGER    PRIMARY KEY,
   P-name   VARCHAR(12)NOT NULL,
   Capacity INTEGER    CHECK (Capacity>0 AND Capacity<= 999),
   Producer VARCHAR(20)DEFAULT  (Boeing),
            CONSTRAINT unique_plane_name UNIQUE (P-Name)
   );
```

2. Inserting Data into the Plane table

```
Insert into plane values (100, 'AIRBUS', 300, ' Air Bus ');
Insert into plane values (101,'B737',250,'Boing');
Insert into plane values (102, 'B737',150,'Boing’);
```

3. Change the capacity of the plane number 101 with the new capacity 220.

```
Update Plane
set Capacity=220
where P-num=101;
```

4. Removal of planes with a capacity less than 200.

```
Delete from plane
where Capacity <200;
```

Exercise 2

Consider the relational schema for loan management in a bank.

Customer (Customer-Num, Last-Num, First-Num, City-Customer)

Account (Account-Num, Customer -Num, Branch-Num, Balance)

Branch (Branch-Num, Branch-Name, Branch-City)

Loan (Employee-Num, Customer, Branch-Num, Amount)

1. **Creating all tables with PK (primary Key) and FK (foreign Key) integrity constraints**

- **Customer Table**

```
CREATE TABLE Customer (
    Customer_Num INT PRIMARY KEY,
    Last_Num VARCHAR(20),
    First_Num VARCHAR(20),
    City_Customer VARCHAR(30)
);
```

- **Branch Table**

```
CREATE TABLE Branch (
    Branch_Num INT PRIMARY KEY,
    Branch_Name VARCHAR(30),
    Branch_City VARCHAR(30)
);
```

- **Account Table**

```
CREATE TABLE Account (
  Account_Num INT PRIMARY KEY,
  Customer_Num INT,
  Branch_Num INT,
  Balance DECIMAL(10,2),
  FOREIGN KEY (Customer_Num) REFERENCES Customer(Customer_Num),
  FOREIGN KEY (Branch_Num) REFERENCES Branch(Branch_Num)
  );
```

- **Loan Table**

```
CREATE TABLE Loan (
  Employee_Num INT,
  Customer_Num INT,
  Branch_Num INT,
  Amount DECIMAL(10,2),
  PRIMARY KEY (Employee_Num, Customer_Num, Branch_Num),
  FOREIGN KEY (Customer_Num) REFERENCES Customer(Customer_Num),
  FOREIGN KEY (Branch_Num) REFERENCES Branch (Branch_Num)
  );
```

2. **Insert one sample record per table**

- **Insert into Customer**

```
INSERT INTO Customer (Customer_Num, Last_Num, First_Num,
City_Customer) VALUES (10, 'Chedari', 'Amine', 'Annaba');
```

- **Insert into Branch**

```
INSERT INTO Branch (Branch_Num, Branch_Name, Branch_City)
VALUES (101, 'Downtown Branch', 'Algiers');
```

- **Insert into Account**

```
INSERT INTO Account (Account_Num, Customer_Num, Branch_Num,
Balance) VALUES (1001, 10, 101, 7500.00);
```

- **Insert into Loan**

```
INSERT INTO Loan (Employee_Num, Customer_Num, Branch_Num, Amount)
VALUES (5001, 10, 101, 15000.00);
```

3. **The SQL commands to satisfy the users' requirements for inserting data into the database:**

BRANCH: (1, 'BRANCH1', 'Guelma'); CLIENT: (1, 'Selamoui', 'Mahmoud', 'Annaba')
ACCOUNT:(1, 2, 3, 25000); LOAN (1, 1, 1, 60000)

- **Insert into BRANCH**

```
INSERT INTO Branch (Branch_Num, Branch_Name, Branch_City)
VALUES (1, 'BRANCH1', 'Guelma');
```

- **Insert into CUSTOMER**

```
INSERT INTO Customer (Customer_Num, Last_Num, First_Num,
City_Customer) VALUES (1, 'Selmaoui', 'Mahmoud', 'Annaba');
```

- **Insert into ACCOUNT**

```
INSERT INTO Account (Account_Num, Customer_Num, Branch_Num,
Balance) VALUES (1, 2, 3, 25000);
```

NOTE: Customer_Num = 2 and Branch_Num = 3 must already exist in Customer and Branch tables. Otherwise, this will violate foreign key constraints

- **Insert into LOAN**

```
INSERT INTO Loan (Employee_Num, Customer_Num, Branch_Num, Amount)
VALUES (1, 1, 1, 60000);
```

4. **Add a strictly positive constraint (>) for Amount.**

This specification can be done by adding a new check constraint to the attribute Amount.

```
CREATE TABLE Loan (
   Employee_Num INT,
   Customer_Num INT,
   Branch_Num INT,
   Amount DECIMAL(10,2) CHECK (Amount > 0),
   PRIMARY KEY (Employee_Num, Customer_Num, Branch_Num),
   FOREIGN KEY (Customer_Num) REFERENCES Customer(Customer_Num),
   FOREIGN KEY (Branch_Num) REFERENCES Branch(Branch_Num)
  );
```

5. **Change the Null value of the amounts to 0.**

```
UPDATE Loan
SET Amount = 0
WHERE Amount IS NULL;
```

6. **Change the Client Cities to lowercase.**

```
UPDATE Customer
SET City_Customer = LOWER(City_Customer);
```

7. Increase the balance of all clients by "0.5%"

```
UPDATE Account
SET Balance = Balance * 1.005;
```

8. Delete the attribute "City-Customer" from the Customer table and add the email attribute.

- **To remove the attribute "City-Customer" from the customer table**

```
ALTER TABLE Customer
DROP COLUMN City_Customer;
```

This will **permanently remove** the `City_Customer` column and its data.

- **To add the new the attribute "email" to the customer table**

```
ALTER TABLE Customer
ADD Email VARCHAR(50);
```

Exercise 3

Student (Student ID, Name, Address)
Course (Course ID, Day, Time)
Assign (Course ID, Room ID)
Evaluate (Course ID, Student ID, Mark)

Expressing the SQL queries for each requirement.

- **Q1: List of students with their addresses.**

```
SELECT Name, Address
FROM Student;
```

- **Q2: The teaching days and times for the DB course**

```
SELECT Day, Time
FROM Course
WHERE Course_ID = 'DB';
```

- **Q3: List of the identifiers of students having obtained more than the average in the DB course.**

```
SELECT Student_ID
FROM Evaluate
WHERE Course_ID = 'DB' AND Mark >= 10
```

Exercise 4

Consider the database schema for the company's employees management of **exercise 2 in series 3**.

EMPLOYEE (E-Num, E-Name, Profession, Hire-Date, Salary, Commission, Dep-Num)
DEPARTMENT (Dep-Num, Dep-Name, Director, City-D)

The SQL queries for expressing the required specifications.

- **Q1: List of engineers (Number, Name) hired before 01/01/2020 with their commission.**

```
SELECT E_Num, E_Name, Commission
FROM EMPLOYEE
WHERE Profession ='Engineer' AND Hire_Date < DATE '2020-01-01';
```

- **Q2: All tuples of EMPLOYEE.**

```
SELECT *
FROM EMPLOYEE;
```

- **Q3: The hire dates of the technicians.**

```
SELECT Hire_Date
FROM EMPLOYEE
WHERE Profession = 'Technician';
```

- **Q4: The name and profession of employee number 10.**

```
SELECT E_Name, Profession
FROM EMPLOYEE
WHERE E_Num = 10;
```

- **Q5: The employees' professions (after eliminating redundant tuples).**

```
SELECT DISTINCT Profession
FROM EMPLOYEE;
```

- **Q6: The names of the employees and their departments.**

```
SELECT E-Name, D-Name
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.Dep-Num = DEPARTMENT.Dep-Num;
```

- **Q7: The numbers of the employees working at ANNABA.**

```
SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYE.Dep-NUM=DEPARTMENT.Dep-Num AND City-D =’ANNABA’;
```

- **Q8: Give the names of the managers of departments 1 and 3.**

```
SELECT Dep-Num, E-Name
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.Dep-Num IN (1,3) AND Director = Emp-Num;
```

- **Q9: The names of the employees working in a department with at least one engineer.**

```
SELECT E2.E-Name
FROM EMPLOYEE E1, EMPLOYEE E2
WHERE E1.Dep-Num = E2.Dep-Num
AND E1.Profession = ’ engineer’;
```

Exercise 5

Consider the relational schema for department management consisting of the two relations.

- EMPLOYEES (Emp_Num, Emp_Name, Position, Salary)
- DEPARTMENTS (Dep_Num, City)

Expressing the required queries with SQL.

1. The name, position, and salary of the employee(s) with the highest salary.

```
SELECT Emp_Name, Position, Salary
FROM EMPLOYEES
WHERE Salary = (SELECT MAX(Salary) FROM EMPLOYEES);
```

The subquery in parentheses is evaluated first and calculates the maximum employee salary. The main query will loop through the rows in the EMPLOYEES table and compare each employee's salary with the result of the subquery. If there is a tie, the employee's name, position, and salary will be included in the final result.

2. The number of different positions held in department 30.

```
SELECT COUNT(DISTINCT Position) AS Num_Positions
FROM EMPLOYEES
WHERE Dep_Num = 30;
```

The **DISTINCT** keyword allows you to count the same value only once. The returned number will be the number of different values of the Function attribute.

3. The average salary per department.

```
SELECT Dep_Num, AVG(Salary) AS Avg_Salary
FROM EMPLOYEES
GROUP BY Dep_Num;
```

If we want to include the **department city name** from the `DEPARTMENTS` table as well, we can use a `JOIN` command as follows:

```
SELECT D.Dep_Num, D.City, AVG(E.Salary) AS Avg_Salary
FROM EMPLOYEES E
JOIN DEPARTMENTS D ON E.Dep_Num = D.Dep_Num
GROUP BY D.Dep_Num, D.City;
```

4. The average annual salary (12 months) of all employees for each department except those with the position of 'MANAGER' or 'PRESIDENT'.

```
SELECT Dep_Num, AVG(Salary * 12) AS Avg_Annual_Salary
FROM EMPLOYEES
WHERE Position NOT IN ('MANAGER', 'PRESIDENT')
GROUP BY Dep_Num;
```

If we want to include the **department city name** from the `DEPARTMENTS` table as well, we can use a `JOIN command, as follows`:

```
SELECT D.Dep_Num, D.City, AVG(E.Salary *12)AS Avg_Annual_Salary
FROM EMPLOYEES E
JOIN DEPARTMENTS D ON E.Dep_Num = D.Dep_Num
WHERE E.Position NOT IN ('MANAGER', 'PRESIDENT')
GROUP BY D.Dep_Num, D.City;
```

5. **The numbers of departments with more than 10 employees.**

```
SELECT COUNT(*) AS Num_Departments
FROM (
    SELECT Dep_Num
    FROM EMPLOYEES
    GROUP BY Dep_Num
    HAVING COUNT(*) > 10
) AS DeptCounts;
```

The `COUNT(*)` command calculates the tuples of the subsets created by `GROUP BY`. The tuples of each subset created have the same value of the attribute that is used in the `GROUP BY` (*i.e. here* `Dep_Num`). The result of this query will be a relation partitioned into groups of tuples having the same value of the `Dep_Num` attribute. To each of these groups will be applied the qualification of the `HAVING` clause, i.e. `COUNT(*) > 10`. If the group satisfies the condition (*contains more than 10 tuples*), it will be included in the result through its `Dep_Num` attribute acting as a common denominator for the tuples in the group.

6. **The different positions held in all departments and the average salary per position.**

```
SELECT Position, AVG(Salary) AS Avg_Salary
FROM EMPLOYEES
GROUP BY Position;
```

Note: Using `AS` allows naming the column created using the `AVG` function.

7. **The different functions performed in all departments having an average salary per function greater than 10,000.**

```
SELECT Position, AVG(Salary) AS Avg_Salary
FROM EMPLOYEES
GROUP BY Position
HAVING AVG(Salary) > 10000;
```

8. **List the employees in Department 10 in descending order of salary.**

```
SELECT Emp_Name, Position, Salary
FROM EMPLOYEES
WHERE Dep_Num = 10
ORDER BY Salary DESC;
```

9. **List of the employees sorted alphabetically by function, and within each function sorted by descending salary.**

```
SELECT Emp_Name, Position, Salary
FROM EMPLOYEES
ORDER BY Position ASC, Salary DESC;
```

Bibliographie

1. Date, C. J. *An Introduction to Database Systems*, 8th Edition, Pearson, 2003. Addison Wesley. ISBN: 0321197844.
2. Georges Gardarin (2003), *Bases de données*. 5ième édition
3. Eric GODOC et Anne-Christine Bisson (2017). *SQL les fondamentaux du langage*. Edition ENI.
4. Jean-Luc Hainaut. *(2010), Bases de données Concepts, utilisation et développement* - Collection Sciences Sup, DUNOD.
5. Bouzeghoub, Mokrane. (2006). *Les bases de données et l'ingénierie des modèles. In l'ingénierie dirigée par les modèles, Au-delà du MDA* (Lavoisier).
6. Forta, B. *SQL in 10 Minutes, Sams Teach Yourself*, 6th Edition, Pearson, 2019.
7. Ramakrishnan, R., & Gehrke, J. *Database Management Systems*, 3rd Edition, McGraw-Hill, 2003.
8. Elmasri, R., & Navathe, S. B. *Fundamentals of Database Systems*, 7th Edition, Pearson, 2016.
9. Silberschatz, A., Korth, H. F., & Sudarshan, S. *Database System Concepts*, 7th Edition, McGraw-Hill, 2019.
10. Celko, J. SQL for Smarties: *Advanced SQL Programming*, 5th Edition, Morgan Kaufmann, 2014.
11. Beaulieu, A. *Learning SQL*, 3rd Edition, O'Reilly, 2020.
12. Winand, M. *SQL Performance Explained*, 2nd Edition, 2012.
13. Connolly, T., & Begg, C. *Database Systems*, 6th Edition, Pearson, 2014.
14. Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases.* Addison-Wesley.
15. Coronel, C., & Morris, S. (2019). *Database Systems: Design, Implementation, & Management*, 13th Edition. Cengage Learning.
16. Harrington, J. L. (2016). *SQL Clearly Explained* (4th ed.). Morgan Kaufmann.
17. Celko, J. (2010). *SQL Programming Style*. Morgan Kaufmann.
18. Tardieu, H., & Rochfeld, A. (1996). *MERISE 2 : Ingénierie des systèmes d'information.* Éditions d'Organisation.
19. Codd, E. F. (1971). *Further Normalization of the Data Base Relational Model.* In *Data Base Systems*, Courant Computer Science Symposia Series, Prentice Hall.