

# ON-LINE DISTANCE COURSE ON DATABASES

## Course Objectives:

The proposed curriculum addresses key issues in the theory of Databases:

- The conceptual model of databases includes the main tasks that solve the problems set;
- The logical model of the database includes the logical connections between the different data, which are the basis of the developed database;
- A physical model of the databases is their physical realization (location, links and management of information);

The **objective** of the course is to present an introduction to database management systems, with an emphasis on how to organize, maintain and retrieve - efficiently, and effectively - information from a DBMS.

The course considers different database models, but emphasizes the understanding of the fundamentals of relational systems including data models, database architectures, and database manipulations.

The course also includes algorithms for query processing and optimization and some of the application databases in different areas of bioinformatics.

# Learning Outcomes:

At the conclusion of the course, the student **be able** to:

- Produces an Entity-Relationship model from a realistic problem specification.
- Describes the conceptual schema of a database.
- Describes the physical schema of a database.
- Uses formal design techniques to produce a database schema.
- Applies normalization techniques.
- Prepares logical construction.
- Designs and applies database from the logical schema model.
- Manages a designed database.
- Arranges database using Relational algebra.
- Organizes database using SQL.
- Discusses the relative merits of the relational environment.
- Describe the fundamental elements of relational database management systems.
- Explain the basic concepts of relational data model, entity-relationship model, relational database design, relational algebra and SQL.
- Work as a valuable member of a database design and implementation team.



## STRUCTURE OF THE ON-LINE DISTANCE COURSE ON DATABASES

COURSE - 140 hours

MODULES – 5 modules

TOPICS – 21 topics

LESSONS – 47 lessons

		COURSE	Duration	Comment
		DATABASES	<b>140</b> hours	
		Title of the module/ topic/ lesson		
MODULE (GROUP OF TOPICS) (5 modules)	<b>1.</b>	<b>Theoretical basis of Databases</b>	<b>24</b>	
	Topics included in the module	1.	<b>Basic concepts</b>	<b>6</b>
		Lessons	1. Introduction to database	3
			2. Introduction to database management system (DBMS)	3
		2.	<b>Logical and physical bases of databases</b>	<b>6</b>
		Lessons	1. Logical foundations of databases	3
			2. Physical foundations of databases	3
		3.	<b>Data models</b>	<b>6</b>
		Lessons	1. What is a Database Model. Types of data models.	3
			2. Database Languages in DBMS: Data description languages. Data manipulation languages.	3
		4.	<b>Basic search methods. File types.</b>	<b>6</b>
		Lessons	1. Basic search methods	3
			2. Types of database files	3
	<b>2.</b>	<b>Relational approach in databases</b>	<b>30</b>	
	Topics included	1.	<b>Relational model - basic concepts, relational schemes</b>	<b>6</b>
		Lessons	1. Relational approach. Relational model	3
			2. Relational algebra	3

	in the module	2.	<b>Relational languages</b>		<b>6</b>	
		Lessons	1.	Relational languages. Types of Relational Languages.	3	
			2.	SQL Relational language. Data selection. Built-in functions. Data updating.	3	
		3.	<b>Relational systems</b>		<b>6</b>	
		Lessons	1.	Basic characteristics and classification of relational systems	3	
			2.	Relational schema analysis. Functional dependencies	3	
		4.	<b>Development of a sample database. Normalization.</b>		<b>6</b>	
		Lessons	1.	Normalization of relational schemas. Normal forms.	3	
			2.	Decomposition and synthesis of relational schemas. Normalization algorithm by decomposition. Algorithm for synthesis of relational schemas.	3	
		5.	<b>Object-oriented database systems</b>		<b>6</b>	
		Lessons	1.	The essence of objects. Basic concepts in the object-oriented approach.	3	
			2.	Object-oriented database management systems. Architecture.	3	
	3.	<b>Algorithms and their applications in databases for query optimization</b>			<b>38</b>	
	Topics included in the module	1.	<b>Introduction to algorithms</b>		<b>8</b>	
		Lessons	1.	The role of algorithms in computation. Algorithms as a technology.	2	
			2.	Design and analysis of algorithms	3	
			3.	Complexity of algorithms. Types of complexity and their estimation	3	
		2.	<b>Strategies in Algorithm Design</b>		<b>12</b>	
		Lessons	1.	Divide and Conquer Paradigm in Algorithms	3	
			2.	Dynamic programming	3	
			3.	Heuristic and probability algorithms	3	
			4.	Greedy algorithms. Examples	3	
		3.	<b>Sorting algorithms</b>		<b>6</b>	

		Lessons	1	Sorting algorithms - part I (Insertion sort, Selection sort, Bubble method).	3		
			2	Sorting algorithms - part II (Linear time sort - Quick sort. Merge sort. Heap sort.)	3		
		4	<b>Graph algorithms</b>			<b>12</b>	
		Lessons	1.	Introduction to graph algorithms		3	
			2.	Tree Cover of Graphs		3	
			3.	Shortest Path Algorithms		3	
			4.	Maximum flow in graph		3	
		<b>4.</b>	<b>Biological Databases</b>			<b>24</b>	
		Topics included in the module	1.	<b>Amino acids, peptides and proteins</b>			<b>6</b>
	Lessons		1.	Amino acids		3	
			2.	Peptides and proteins		3	
	2.		<b>Primary and Secondary Databases of Protein. 3D structure protein databases.</b>			<b>6</b>	
	Lessons		1.	Primary and Secondary Databases of Protein		3	
			2.	3D structure protein databases		3	
	3.		<b>Gene databases</b>			<b>6</b>	
	Lessons		1.	Types of genome databases		3	
			2.	Application of gene databases		3	
	4		<b>KEGG: Kyoto Encyclopedia of Genes and Genomes – high-level functions and utilities of the biological system</b>			<b>6</b>	
	Lessons		1.	KEGG Database		3	
			2.	KEGG Software		3	
	<b>5.</b>	<b>Practice</b>			<b>24</b>		
	Topics included in the module	1.	<b>Introduction to the computer program MS Access. Use of tables, subforms, filters and reports.</b>			<b>6</b>	
		Lessons	1.	Basic concepts in databases. Introduction to the computer program MS Access.		3	

			2.	Use of tables and subforms. Using filters and reports.	3	
		2.		<b>Maintaining database changes. Ensuring the reliability of information in the database.</b>	<b>6</b>	
		Lessons	1.	Maintaining database changes	3	
			2.	Ensuring the reliability of information in the database	3	
		3.		<b>Creation of a query. Use of queries.</b>	<b>6</b>	
		Lessons	1.	Creation of a query	3	
			2.	Use of queries	3	
		4		<b>Merging of data into one form. Presentation of an effective report.</b>	<b>6</b>	
		Lessons	1.	Merging of data into one form	3	
			2.	Presentation of an effective report	3	

# ON-LINE DISTANCE COURSE ON DATABASES

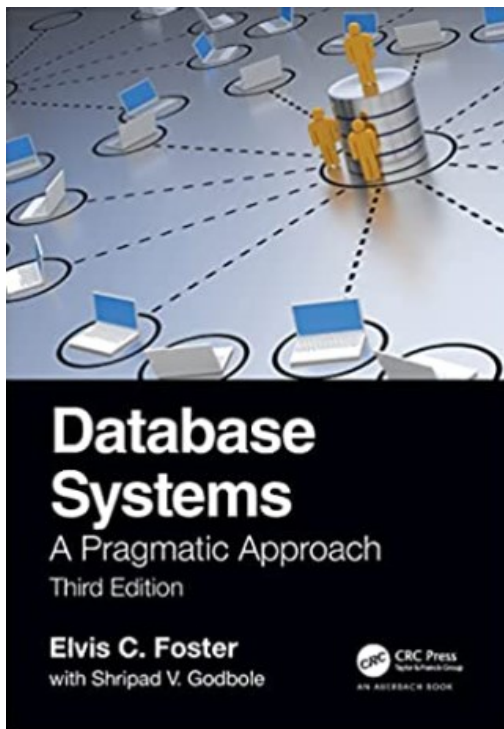
- ❑ **Module 1. Theoretical basis of Databases**
  - ❑ **Topic 1. Basic concepts**
    - ❑ Lesson 1. Introduction to database



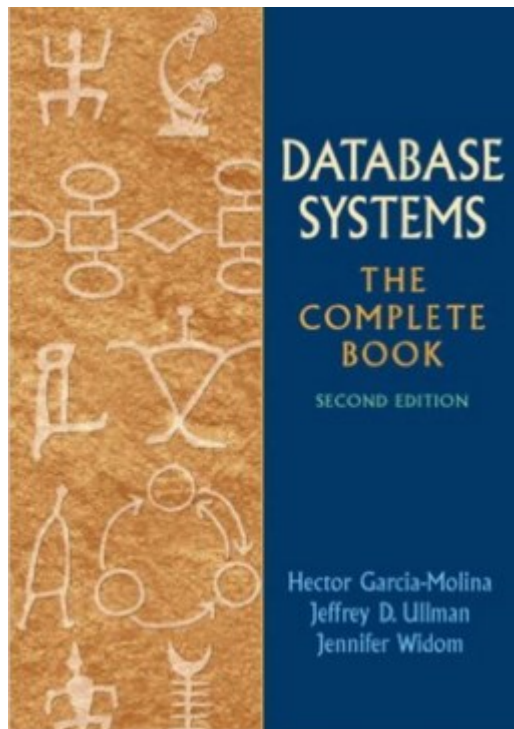
ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# INTRODUCTION TO DATABASE



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



# Introduction

- The term database (DB) appears in the early 1960s, but it is not known who first used it in connection with computer processing of data.
- The ideas of C. Bachmann, reflected in two of his publications from 1964 and 1965, are also fundamental. With them he posed the first problems of a new field in informatics, which is known today as "Database Management".



# Introduction

- At the same time, specialists from other places began to work on database management issues, and large software companies, in their strive for leadership in this field, quickly turned to the program implementation of the first modern database management systems (DBMS).
- A **DBMS** is a powerful tool for creating and efficiently managing large amounts of data. Data must be maintained and stored for as long as necessary. **Lesson 2** will go over DBMS in depth.

# WHAT IS DATABASE

- It is difficult to give a concise and precise definition of the term DB and perhaps this explains the chaos of its various interpretations.
- The continued evolution of the concept itself may also have contributed to the formation of different opinions.
- The perception of the specialists who first considered this question is that:
  - *A DB is a collection of files;*

# WHAT IS DATABASE

- *The file is a set of records;*
- *the record consists of one or more keys and data.*
- This definition is still acceptable to many DBMS developers and users.
- However, it does not depicts the most important property that is claimed for each DB.
- The point is that the DB should be built as an **integrated** set of data.

# WHAT IS DATABASE

- There is a deep meaning in this property of the DB, which is first of all used to express the way the DB is used and managed.
- The fact that a DB is an **integrated** data set means that it contains data for different applications (users), not just for one, as is often seen in practice.
- However it doesn't mean that each user uses the entire data set or that he must know the work of the others.

# WHAT IS DATABASE

- Most commonly, each user is only interested in certain parts of the DB, using their application programs to retrieve and process data, enter new, update or remove existing data in the DB.
- It is also interesting to note that different application programs can use the same subset of DB data.

# WHAT IS DATABASE

- **Consider an example:** In one organization, four computer systems have been set up for the needs of the “Human resources”, “Accounting”, “Research Projects” and “International Cooperation” departments.
- Each of these systems supports one data file. Some of the fields of these files are represented by a table:

# WHAT IS DATABASE

File "Personnel"	File "Salary"
<ol style="list-style-type: none"> <li>Office number</li> <li>Name of employee</li> <li>Department</li> <li>Date of commencement of work</li> <li>Education</li> <li>Speciality</li> </ol>	<ol style="list-style-type: none"> <li>Office number</li> <li>Name of employee</li> <li>Department</li> <li>Duration of employment</li> <li>Base salary</li> <li>Family status</li> </ol>
File "Projects"	File "Business trips"
<ol style="list-style-type: none"> <li>Task <ol style="list-style-type: none"> <li>Number</li> <li>Name</li> </ol> </li> <li>Task manager <ol style="list-style-type: none"> <li>Office number</li> <li>Name of employee</li> <li>Department</li> <li>Position</li> </ol> </li> <li>Period <ol style="list-style-type: none"> <li>Date of beginning</li> <li>Date of ending</li> </ol> </li> <li>Financing organization</li> <li>Periods</li> <li>Lead organization</li> <li>Way of reporting</li> </ol>	<ol style="list-style-type: none"> <li>Office number</li> <li>Name of employee</li> <li>Department</li> <li>Type of business trip</li> <li>Destination</li> <li>Number of days</li> <li>Task</li> </ol>



# WHAT IS DATABASE

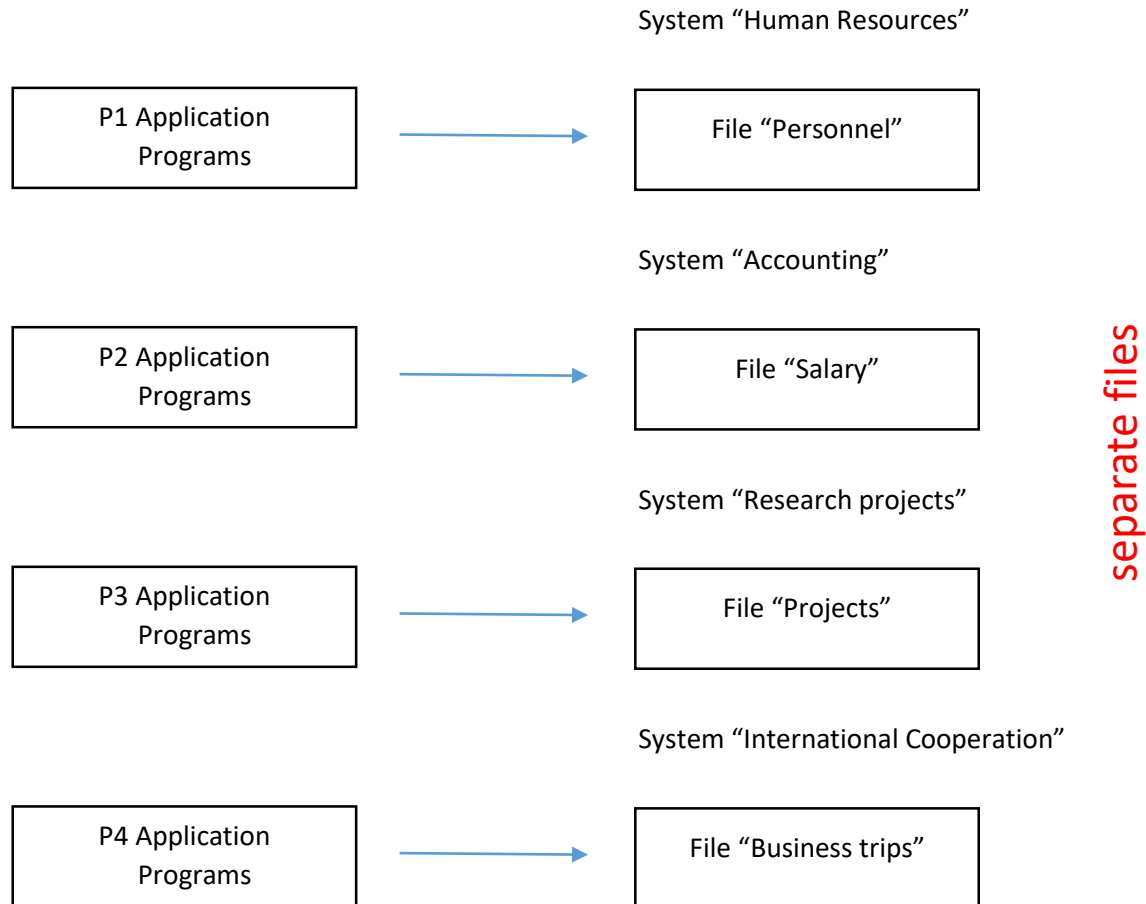
- Quite a few of the fields of each file are contained in other files. Naturally, this leads to inconveniences:

1. **Extra expenditure** of resources for multiple duplication of data entry in different files. For example, if Person A starts work, this fact must be reflected in the “Personnel” and “Salary” files. However, if Employee A subsequently becomes manager of a certain task, some of the data about him in the "Personnel" file must also be entered in the "Project" file.

# WHAT IS DATABASE

2. Changing the date of any employee will require **multiple changes** to individual files. On the one hand, this means that there will be again unnecessary expenditure of funds. On the other hand, making changes to individual files generally leads to inconsistencies in the data of individual systems at some point. It may be that Employee A has left the organization and this is reflected in the Personnel file but not in the Payroll file. It is possible for an employee to change her last name due to marriage and for a period of time she may appear under two different names in the Personnel and Salary files.

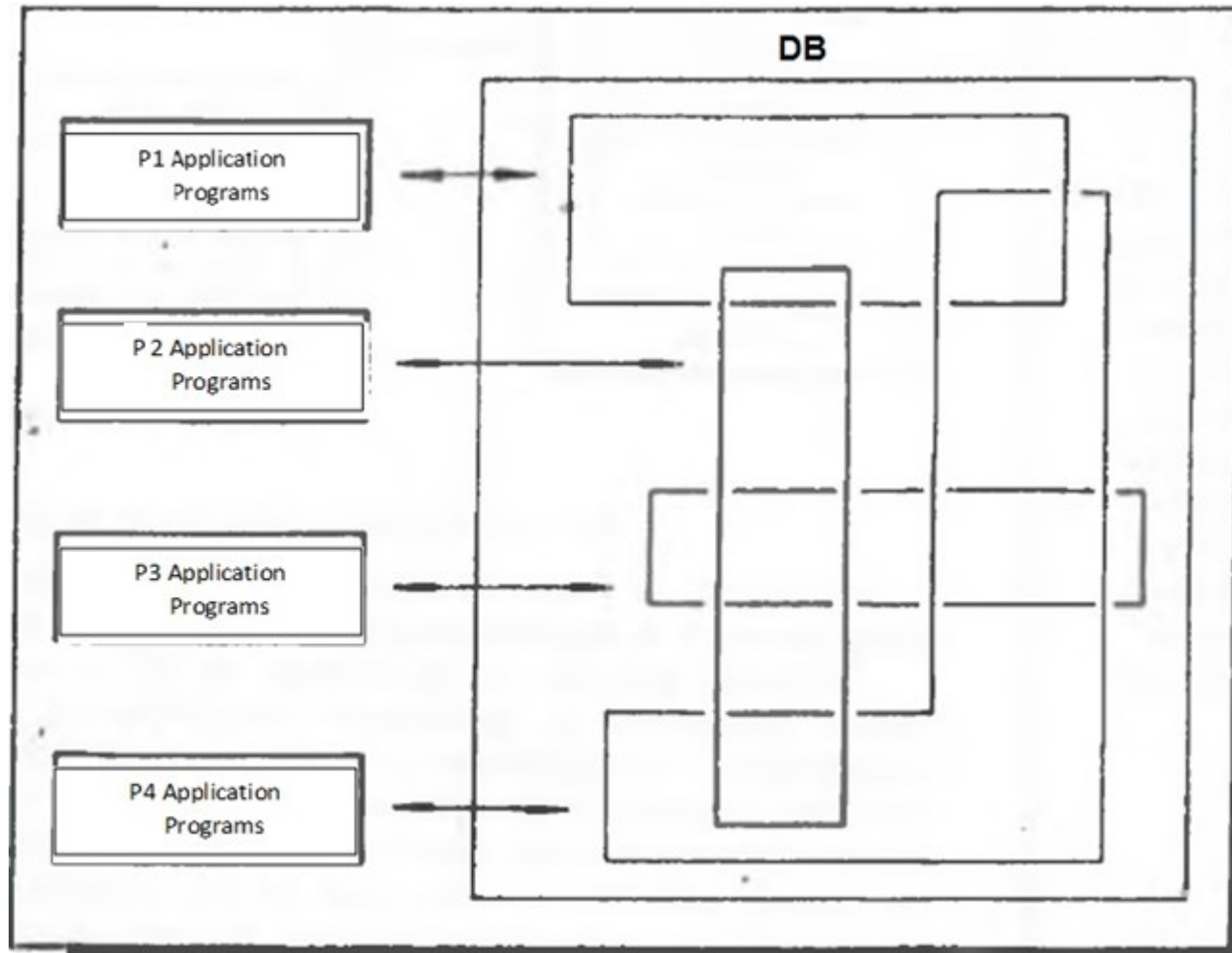
# WHAT IS DATABASE



# WHAT IS DATABASE

- One reasoned solution for curing the above mentioned weaknesses in the four systems (figure above) work is to reorganize their work by **merging (integrating)** the data they store into a single data archive or, as is often said, by creating an integrated DB in the relevant organization.

# WHAT IS DATABASE



integrated set of data

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# WHAT IS DATABASE

- The approach aiming at creating an unified data archive and it's centralized managing has a number of **advantages**.
- The available opportunity to create conditions for continuous expansion and modification of the DB should be highlighted.

# WHAT IS DATABASE

- The point of this approach should be sought in the ability to make extensions and modifications to the DB by simple means, without disrupting the functions of existing application programs, and this should be the most essential consideration for the creation of any DB.
- The DB must be a dynamic and constantly evolving set of data, allowing its applications to change and expand over time. The debate on these issues boils down to the so-called the issue of **data independence**.



# WHAT IS DATABASE

- **Database** - *a set of data structured in a way that allows easy and quick retrieval, review, search and minimizes duplication of information.*
- A characteristic of databases is that the data is independent of the software. This makes them versatile for use both by different programs and in different time periods.

# DATA INDEPENDENCE

- Let's go back to the **example**. Suppose that in application programs **P1** the value of the field “Office Number” is a non-negative integer, while in application programs **P2** the field is described as a character string.
- When merging the data into a single archive, it is natural for a contradiction to arise due to the dependency of the programs on the “Office number” field type.
- There are ways to overcome this contradiction and one of them is to make appropriate changes to the **P1** and **P2** programs.

# DATA INDEPENDENCE

- However, this approach is not satisfactory because not only does it involve additional expenditure, but new changes in the data will again require changes in the relevant programs.
- The independence between programs and data must be two-sided.
- This means that modifications related to the organization of data in external memory should not lead to modifications in application programs and, contrary, modifications in user programs related to the description of the data being used should not affect the way data is stored in the DB.

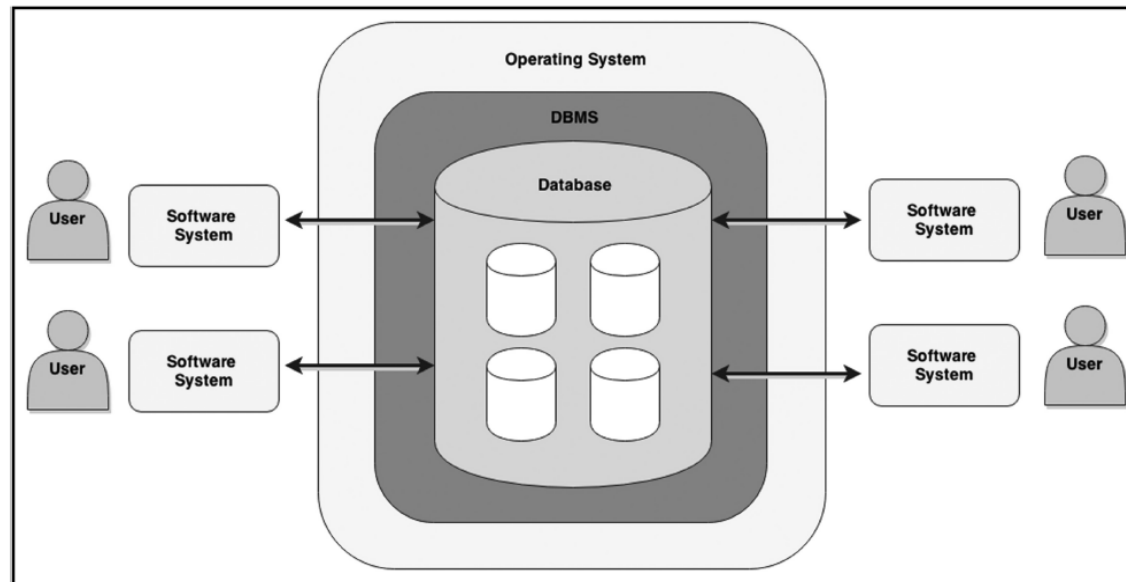
# DBS: Definitions and Rationale

- A database system (DBS) is a computerized record-keeping system with the overall purpose of maintaining information and making it available whenever required. The database typically stores related data in a computer system.
- Components of a **DBS** include:
  - Hardware and operating system
  - DBMS
  - Database
  - Related software systems and/or applications
  - Users — including technical users and end users

# DBS: Definitions and Rationale

- Database users communicate with the software systems/applications, which in turn communicate (through the programming interface) with the DBMS. The DBMS communicates with the operating system (which in turn communicates with the hardware) to store data in and/or extract data from the database, which is illustrated in Figure:

Simplified illustration of a DBS



ERASMUS+

# DBS: Definitions and Rationale

- Databases are essential to software engineering; many software systems have underlying databases that are constantly accessed, though in a manner that is transparent to the end-user.

<i>Software Category</i>	<i>Database Need</i>
Operating systems	A sophisticated internal database is needed to keep track of various resources of a computer system including external memory locations, internal memory locations, free space management, system files, and user files. These resources are accessed and manipulated by active jobs. A job is created when a user logs on to the system and is related to the user account. This process (also called a job) can in turn create other jobs, thus creating a job hierarchy. When you consider that in a multi-user environment, there may be several users and hundreds to thousands of jobs, as well as other resources, you should appreciate that underlying an operating system is a very complex database that drives the system.
Compilers	Like an operating system, a compiler has to manage and access a complex dynamic database consisting of syntactic components of a program as it is converted from source code to object code.
Information systems	Information systems all rely on and manipulate related databases, in order to provide mission-critical information for organizations. All categories of information systems are included. Common categories include (but are not confined to) decision support systems (DSS), executive information systems (EIS), management information systems (MIS), Web information systems (WIS), enterprise resource planning systems (ERPS), and strategic information systems (SIS).
Expert systems	At the core of an expert system is a knowledge base containing cognitive data, which is accessed and used by an inference engine, to draw conclusions based on input fed to the system.

... ..

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# DBS: Definitions and Rationale

- Companies that compete in the marketplace need databases to store and manage their mission-critical data and other essential data.
- What would life be like without contemporary database systems? If you know someone who is old enough, ask him/her about such an era of filing cabinets, hand-written records, or typewriter-generated documents. Life was very slow then, but it was the norm.
- There are several primary and secondary objectives of a database system that should concern the computer science.



# DBS: Conclusion

- The primary objectives of a database system include the following:
  - Security and protection — prevention of unauthorized users; protection from inter-process interference
  - Reliability — assurance of stable, predictable performance
  - Facilitation of multiple users
  - Flexibility — the ability to obtain data and effect action via various methods
  - Ease of data access and data change
  - Accuracy and consistency
  - Clarity — standardization of data to avoid ambiguity
  - Ability to service unanticipated requests
  - Protection of the investment — typically achieved through backup and recovery procedures
  - Minimization of data proliferation — new application needs may be met with existing data rather than creating new files and programs
  - Availability — data is available to users whenever it is required

## DB: Conclusion

- In addition to the above, there are some additional objectives that one may argue are just as important:
  - Physical data independence — storage hardware and storage techniques are insulated from application programs
  - Logical data independence — data items can be added or subtracted, or the overall logical structure modified, without affecting existing application programs that access the database
  - Control of redundancy — the general rule is to store data minimally and not replicate that storage in multiple places unless this is absolutely necessary
  - Integrity controls — range checks and other controls must prevent invalid data from entering the system
  - Clear data definition — it is customary to maintain a data dictionary that unambiguously defines each data item stored in the database
  - Suitably user-friendly interface — be it graphical, command based, or menu based
  - Tunable — easily reorganizing the database to improve performance without changing the application programs
  - Automatic reorganization or migration to improve performance

## Questions and exercises:

1. What is a Database and what is its utility?
2. Explain a few advantages of a Database.
3. What is a DBMS and what is its features?
4. What is a DBS and what are its components?



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 1. Theoretical basis of Databases

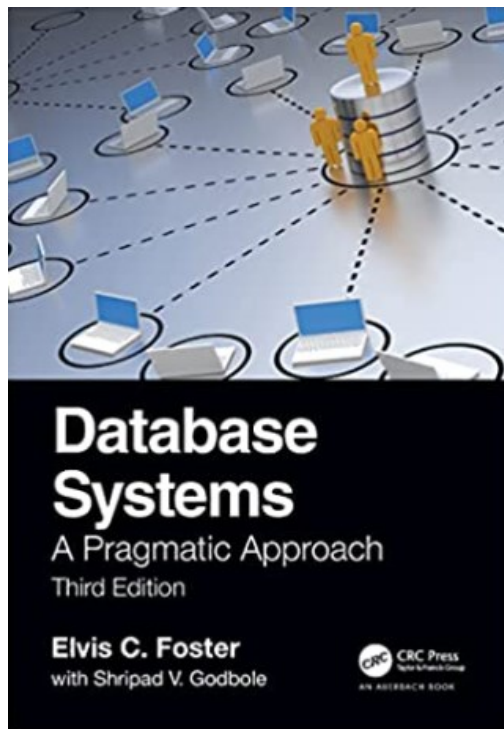
### ❑ Topic 1. Basic concepts

#### ❑ Lesson 2. Introduction to database management system (DBMS)

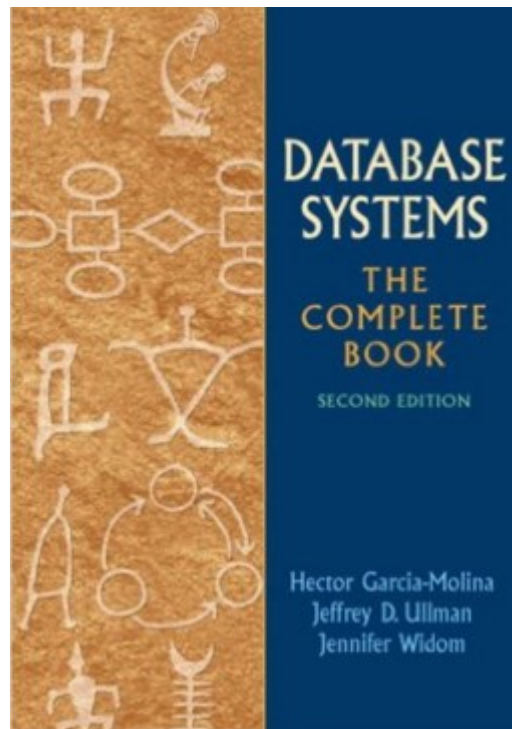


ERASMUS+

# INTRODUCTION TO DATABASE MANAGEMENT SYSTEM (DBMS)



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.



# DBMS

- Any database management system is a software system whose purpose is to create and manage the data organized in a database.
- Management here means searching and processing data and keeping it up to date.
- The last function of the DBMS requires the presence of operations for adding new data, modifying and removing existing data in the DB.
- The proper functioning of a DBMS requires that it provides the DB requirements described in the previous paragraphs:

# DBMS

- separation of the data description from their processing;
- logical and physical independence;
- minimal data redundancy in the DB;
- convenient and with great expressive power user interface;
- efficient processing of requests
- ensuring the integrity of the data in the DB, i. e. ensuring logical consistency in the DB and the appropriate confidentiality of individual users' data.



# DBMS

- From a user perspective, a DBMS must have the following properties:
  - **persistent memory** - data must be stored independently of processes; databases must allow efficient access to very large volumes of data - the access time to the data must be independent of its volume;
  - **program interface** - the DBMS provides a powerful query language for the user or the application program that uses the data;

# DBMS

- From a user perspective, a DBMS must have the following properties:
  - **transaction management** - the DBMS supports simultaneous access to data; it is implemented through multiple processes called transactions; each user who works with the data must initiate such a process; The DBMS maintains transaction isolation - each transaction is **atomic**, i. e. it either completes in full or is rejected in full; if a problem occurs during the execution of a transaction it is canceled, along with any changes it has caused to the database;

# DBMS

- From a user perspective, a DBMS must have the following properties:
  - **data stability** - data must be recoverable in the event of failures or errors; of course, this involves additional reading and writing operations.
- Architecturally, a DBMS must provide the following functions:

# DBMS

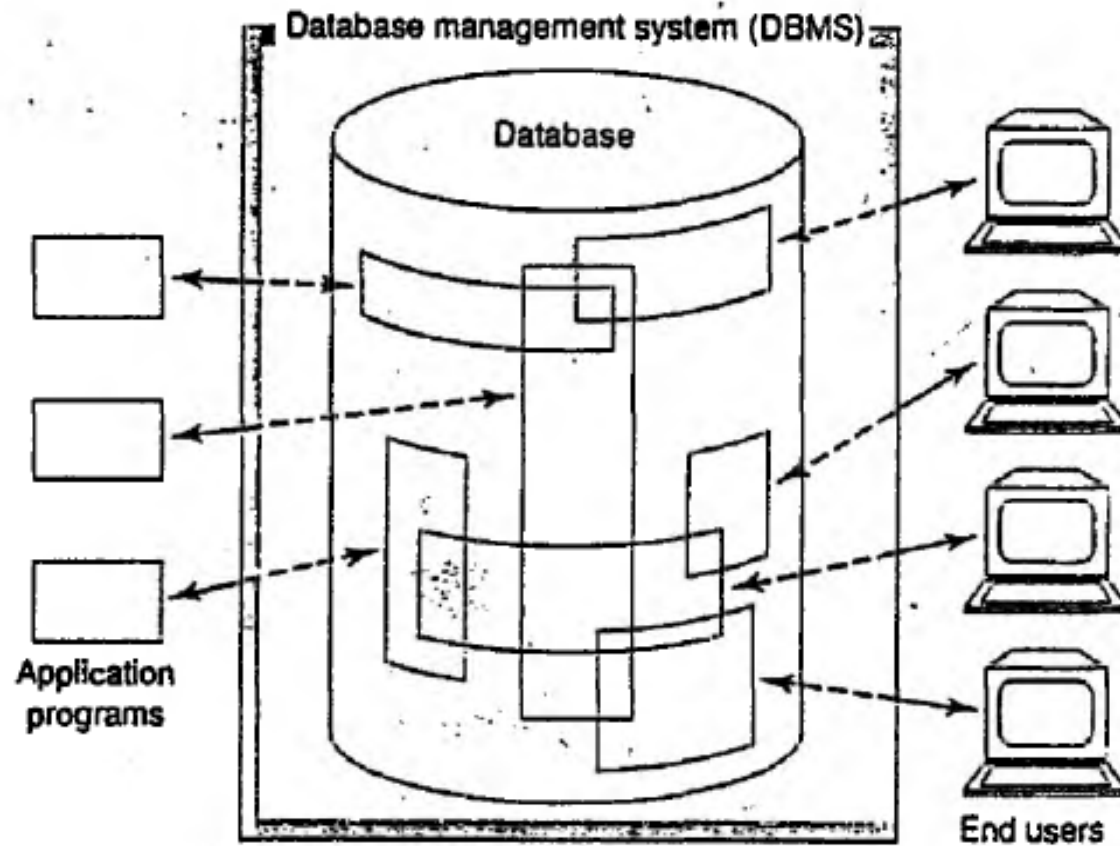
- Architecturally, a DBMS must provide the following functions:
  - The DBMS allows user to create new databases; for this purpose, the user sets the **scheme** of the database; the scheme is a logical data structure and it is described using a special **data definition language (DDL)**; this language is symbolic or graphic;

# DBMS

- Architecturally, a DBMS must provide the following functions:
  - The DBMS must provide the ability to search in the data and to modify the data; this is achieved using a data manipulation language (DML); the query language is a sublanguage of the data manipulation language that is used when searching for data;

# DBMS

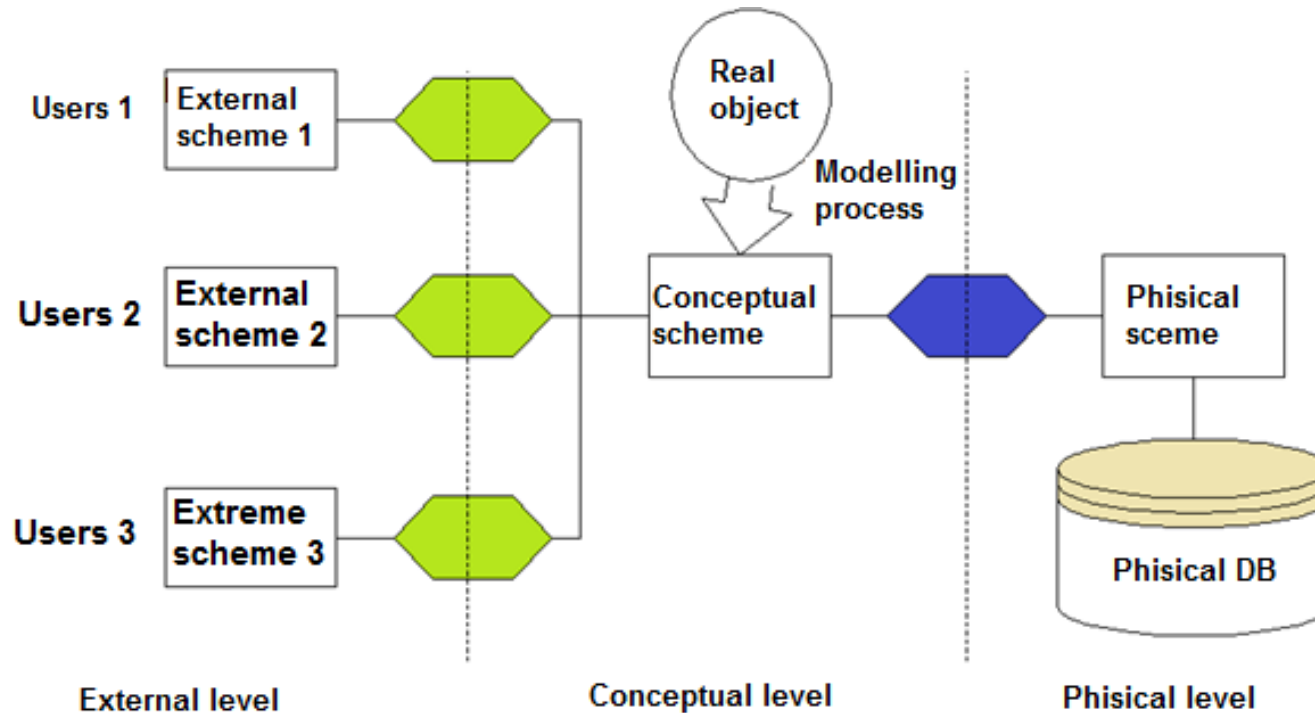
- Architecturally, a DBMS must provide the following functions:
  - The DBMS must store very large volumes of data safely and long-term; storage must ensure efficient searching and modification of data;
  - The DBMS must manage simultaneous access to data by multiple users - they must not interact with each other and destroy the integrity of the data.



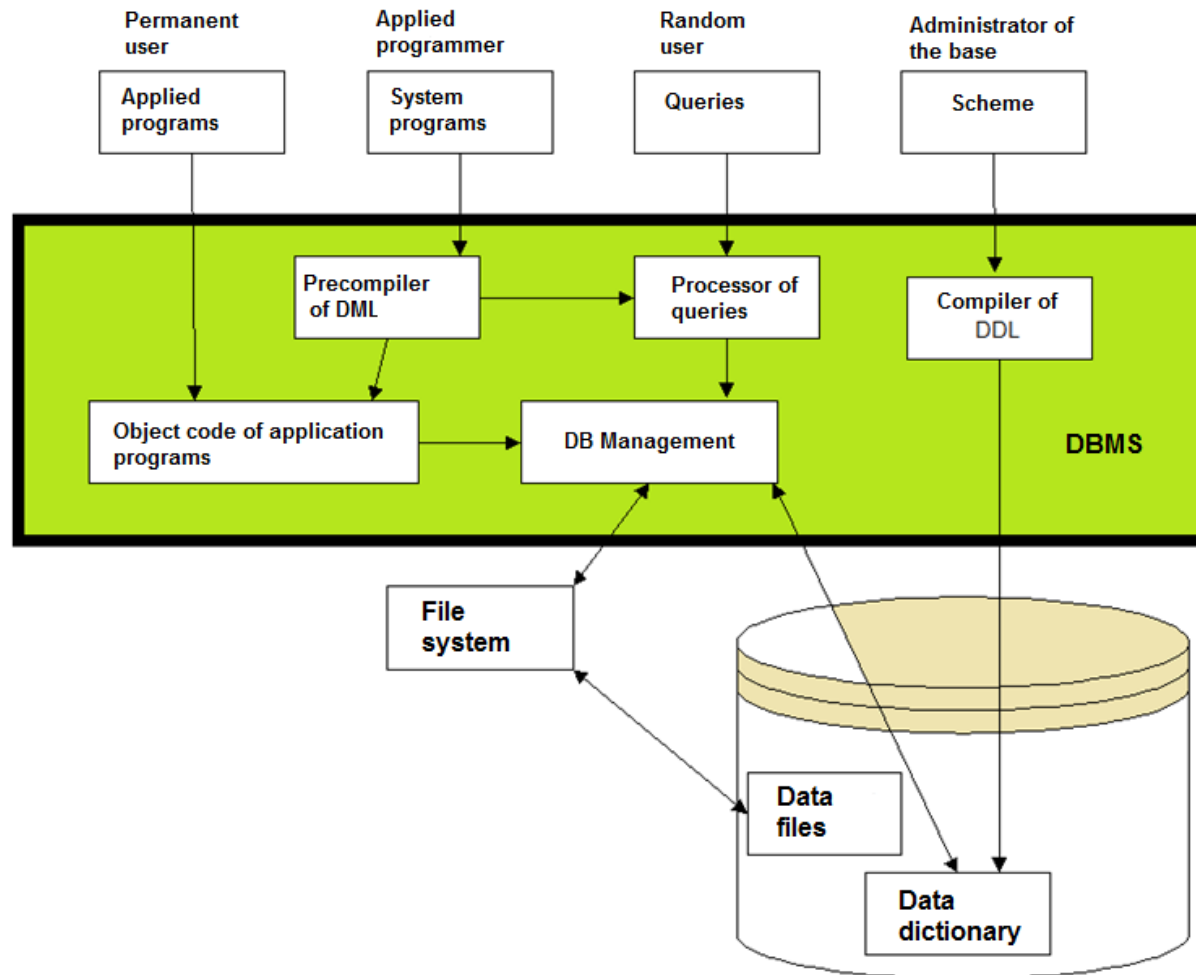
**Simplified picture of a database system**

## *Performance levels of the DB*

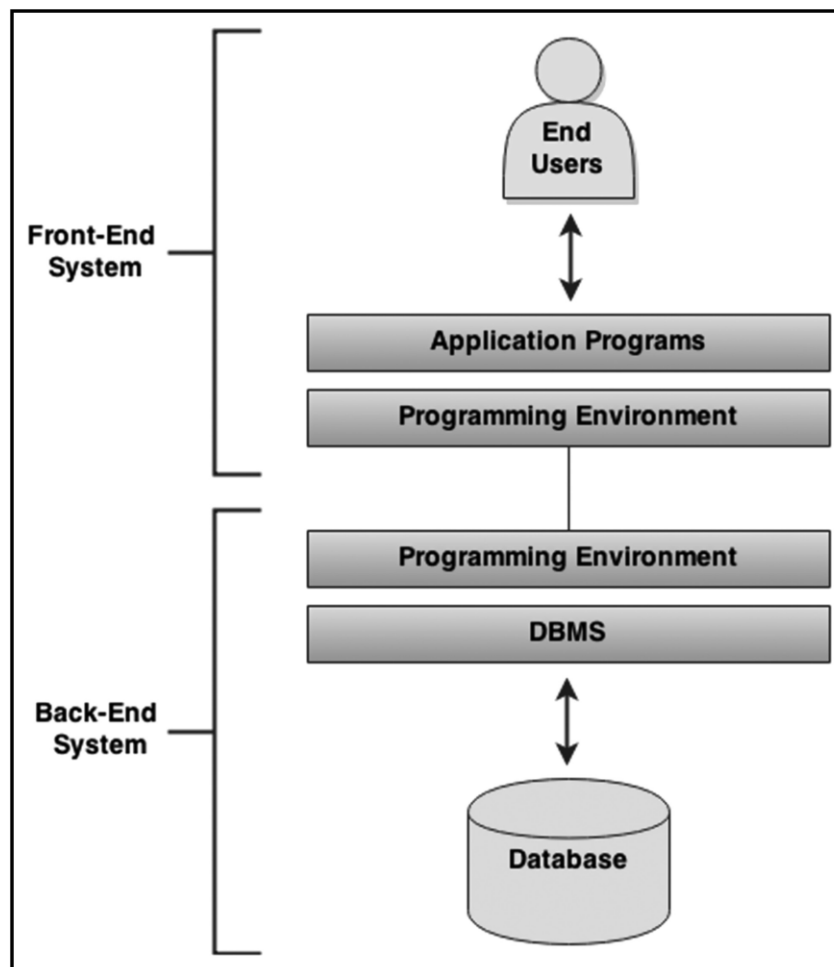
Internal (physical) level  
Conceptual level







## Example of Functional scheme of DBMS



**Front-end and back-end perspectives.**

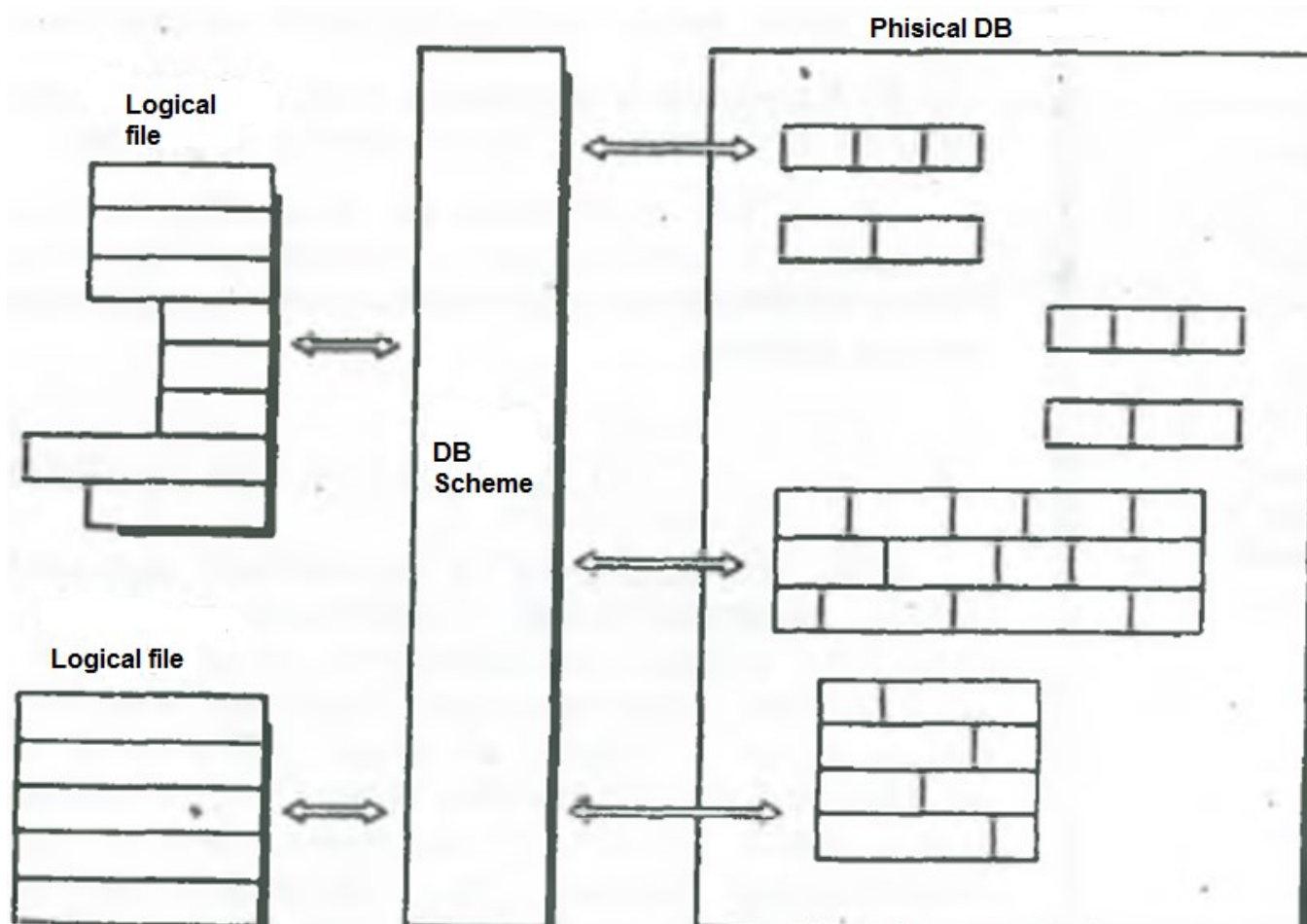
# DBMS

- So, we can assume that the DB is a collection of logical files of individual users, and that the DB is a collection of data files stored in external memory, i. e. we have a logical and a physical DB.
- What is interesting in this case is that both logical and physical DBs can change without one affecting the other.
- To enable this independence in the behavior of the two DBs, an intermediate level of data representation is introduced. It is known by the name conceptual model or conceptual scheme, and most often just a scheme of the DB.

# DBMS

- The scheme is a general logical description of the DB.
- It contains names of objects, descriptions of some of their characteristics, and the relationships that exist between them.
- It is created and kept up-to-date by specialists called by the common name of DB administrator.

# DBMS



# DBMS

- In the presence of a DB scheme, changes made in the logical DB (in the logical files) are reflected in the DB scheme, but do not affect the way data is stored in the physical DB or in other application programs.
- Similarly, changes in the way data is stored and accessed in the physical DB will not cause changes in the logical files, i. e. in the application programs.

# DBMS - Conclusion

- The database management system (DBMS) is the software that facilitates the creation and management of the database. When a user issues a request via some DSL (typically SQL), it is the DBMS that interprets such a request, executes the appropriate instructions, and responds to the request.
- Depending on the nature of the initial request, the response may be relayed (by the DBMS) directly to the end user or indirectly to the end user via an executing application program.

# DBMS - Conclusion

- Through the DBMS, the objectives of the DBS that were mentioned in Lesson 1 are achieved. The primary functions of this very important software system include the following:
  - Data definition (relation, dependencies, integrity constraints, views, etc.)
  - Data manipulation (adding, updating, deleting, retrieving, reorganizing, and aggregating data)
  - Data security and integrity checks
  - Management of data access (including query optimization), archiving, and concurrency
  - Maintenance of a user-accessible system catalog (data dictionary)
  - Support of miscellaneous non-database functions (e.g., utilities such as copy)
  - Programming language support
  - Transaction management (either all changes are made or none is made)
  - Backup and recovery services
  - Communication support (allow the DBMS to integrate with underlying communications software)
  - Support for interoperability including open database connectivity (ODBC), Java database connectivity (JDBC), and other related issues



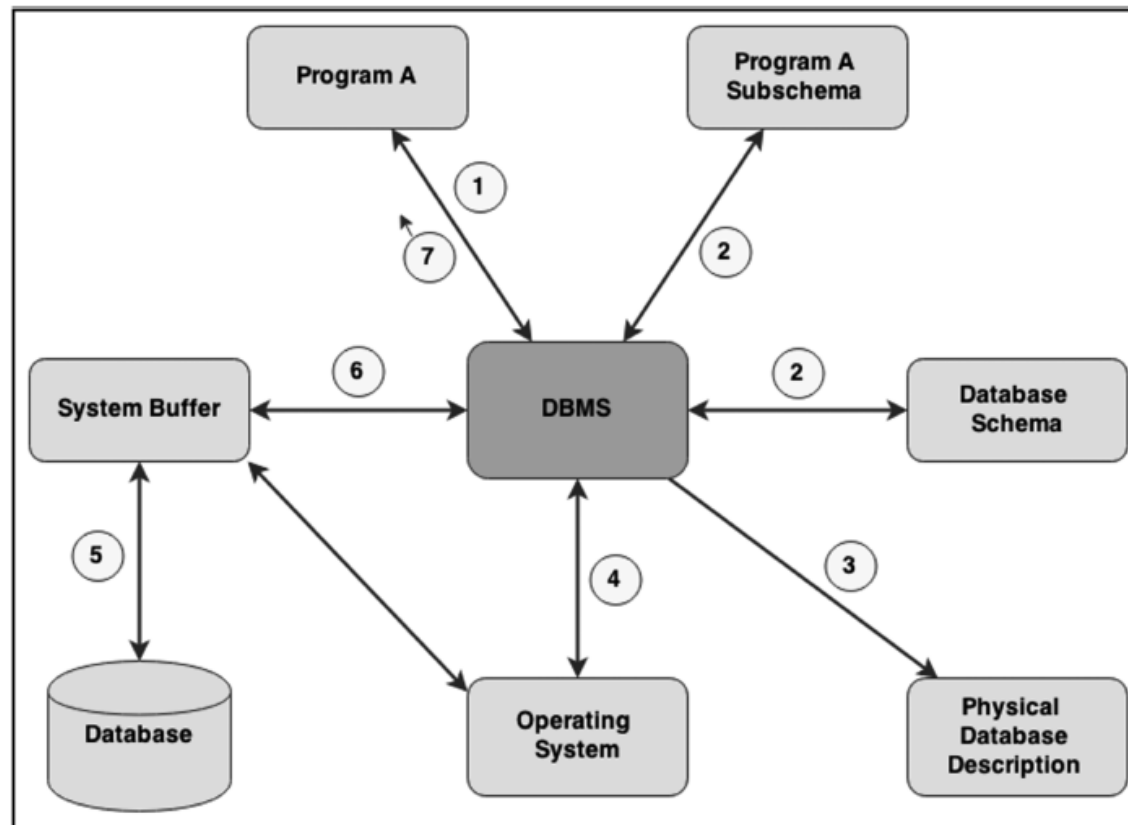
# DBMS - Conclusion

- Optimum efficiency and performance are the hallmarks of a good DBMS. To illustrate the critical role of the DBMS, consider the steps involved when an application program accesses the database:
  - Program-A issues a request to the DBMS (expressed in terms of sub-schema language).
  - DBMS looks at Program-A sub-schema, schema, and physical description (this information is stored in tables).
  - DBMS determines the optimal way to access the data, determining which files must be accessed, which records in the files are needed, and the best method to access DBMS issues instruction(s) (reads or writes) to the operating system.
  - Operating system causes data transfer between disk storage and main memory.
  - DBMS issues move to transfer required fields.
  - DBMS returns control to Program-A (possibly with a completion code).
  - Database Systems.

# DBMS - Conclusion

- Figure bellow provides a graphic representation, but bear in mind that these steps are carried out automatically in a manner that is transparent to the user:

Steps involved when application programs access a database.



ERASMUS+

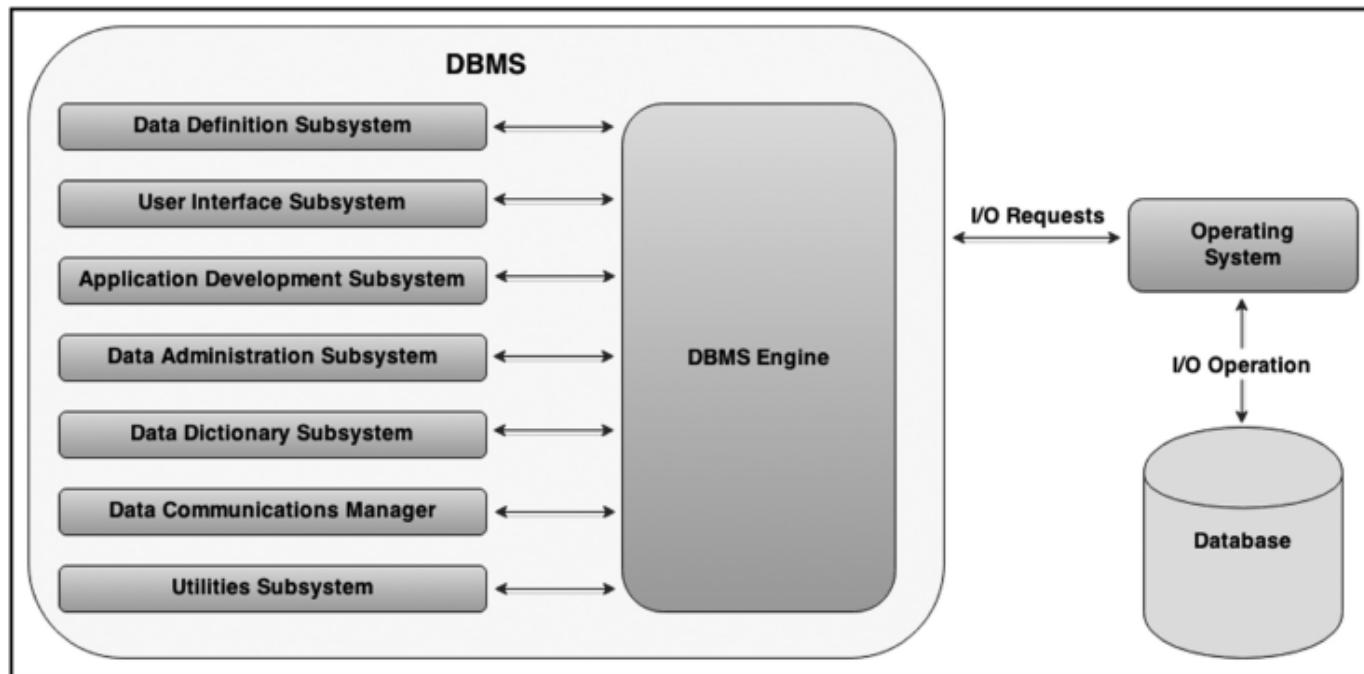
Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# DBMS - Conclusion

- The DBMS is actually a complex conglomeration of software components working together for a set of common objectives. For the purpose of illustration, we may represent the essential components of the DBMS as the following:
- DBMS Engine
- Data Definition Subsystem
- User Interface Subsystem
- Application Development Subsystem
- Data Administration Subsystem
- Data Dictionary Subsystem
- Data Communications Manager
- Utilities Subsystem

# DBMS - Conclusion

- These functional components are not necessarily tangibly identifiable, but they exist to ensure the acceptable performance of the DBMS:



Functional components of a DBMS.

# DBMS - Conclusion

- The DBMS engine is the link between all other subsystems and the physical device (the computer) via the operating system. Some important functions are as follows:
  - Provision of direct access to operating system utilities and programs (e.g., input/output requests, data compaction requests, communication requests, etc.)
  - Management of file access (and data management) via the operating system
  - Management of data transfer between memory and the system buffer(s) in order to effect user requests
  - Maintenance of overhead data and metadata stored in the data dictionary (system catalog)

# Questions and exercises:

1. What is DBMS and what is its utility?
2. Explain a few advantages of a DBMS.
3. What is difference between file system and DBMS ?



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 1. Theoretical basis of Databases

### ❑ Topic 2. Logical and physical bases of databases

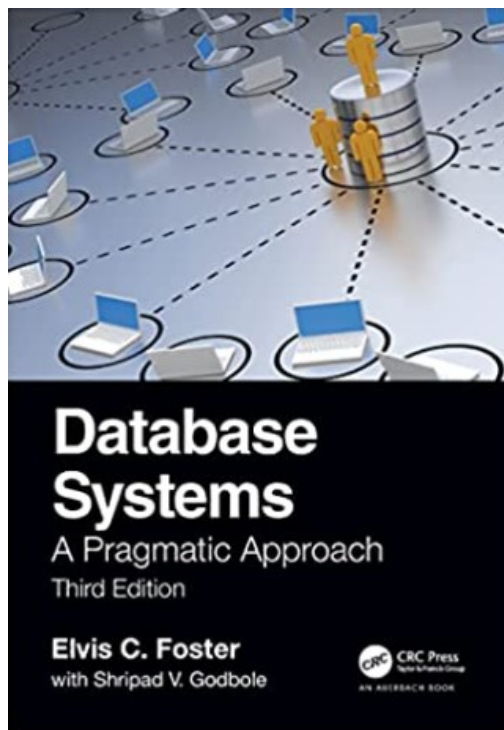
#### ❑ Lesson 1. Logical foundations of databases



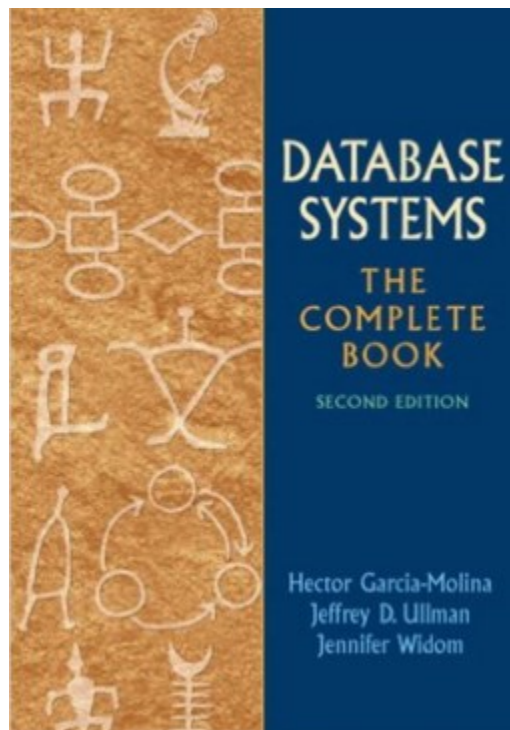
ERASMUS+



# LOGICAL FOUNDATIONS OF DATABASES



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# INTRODUCTION

❑ Database design process has **two** main phases:

❑ **Logical** database design;

❑ **Physical** database design.

LOGICAL DATA MODEL	PHYSICAL DATA MODEL
Model that describes the data as much as possible, without regard to how they will be physical implemented in the database	Model that represents how the actual database is built
Defines the data elements and their relationships	Allows developing the actual database
Data Architects and business analysts create logical data model	Database Administrators and developers create physical data model
The objective of logical data model is to develop a technical map of rules and data structures	The objective of physical model is to implement the actual database
Simpler than the physical data model	Complex than the logical data model
	Visit <a href="http://www.PEDIAA.com">www.PEDIAA.com</a>

ERASMUS+

# Database Design

## □ Logical database design

- Process of constructing a **model of information** used in an enterprise based on a **specific data model** (e.g. **relational**), but **independent** of a particular **DBMS** and other **physical** considerations.

## □ Physical Design

- Process of producing a **description** of the **implementation** of the database on secondary **storage**
  - it describes the **base relations**, **file** organizations, and **indexes** design used to achieve efficient **access** to the data, and any associated **integrity constraints** and **security** measures.

# Purpose of Database Design

- ❑ Structure the data in stable **structures**, called **normalized tables**
  - ❑ Not likely to **change over time**
  - ❑ **Minimal** redundancy
- ❑ Develop a **logical database design** that reflects **actual data requirements**
- ❑ It forms the **base** for a **physical** database design

# Purpose of Database Design

- ❑ Translate a **relational** database model into a **technical file** and **database design** that balances several performance factors
  
- ❑ Choose data **storage** technologies that will **efficiently, accurately** and **securely** process database activities



# Process of Database Design

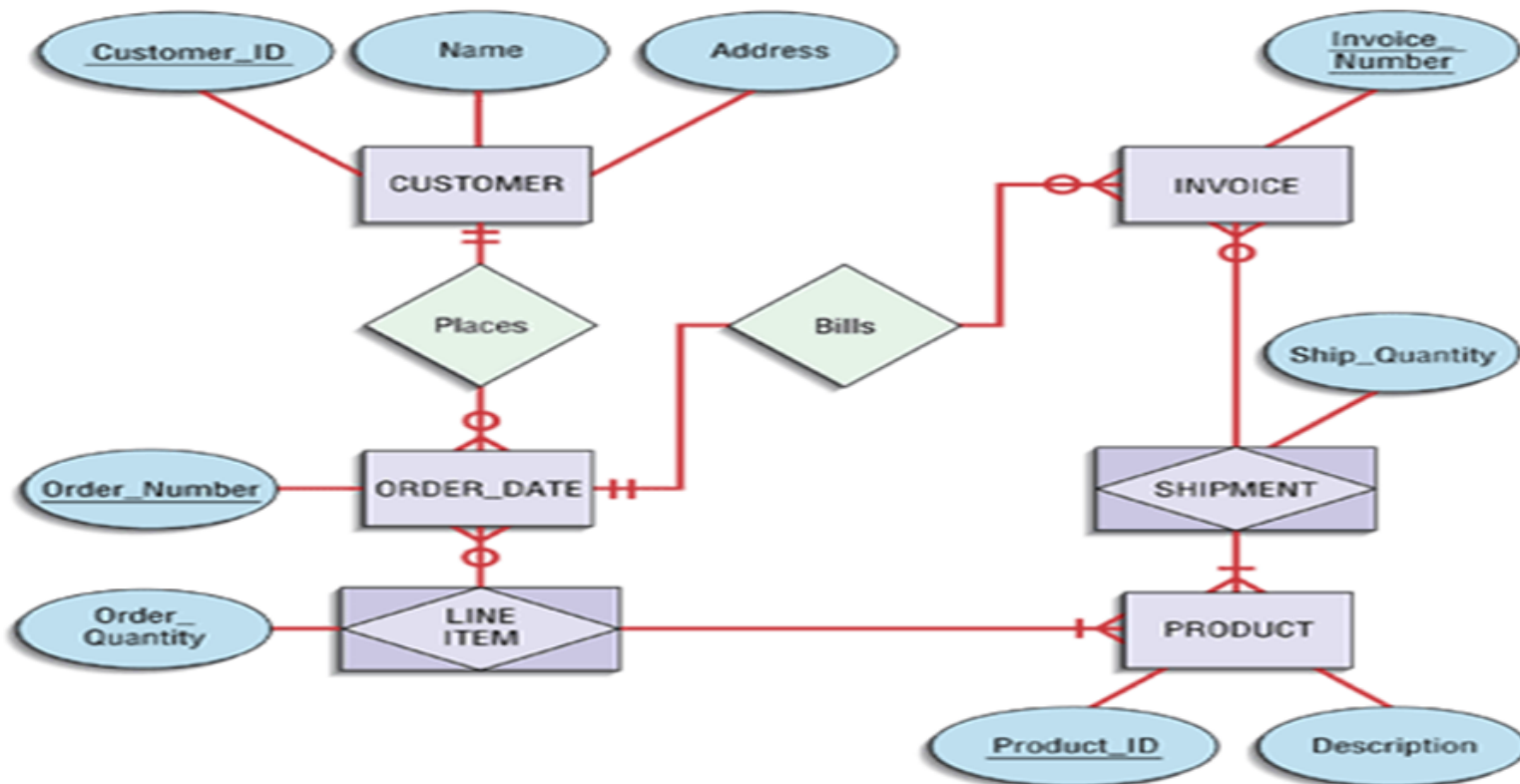
## ☐ Logical Design

- ☐ **Based** upon the **conceptual** data model

- ☐ **4** key steps

1. Develop a **logical** data model for each known **user interface** for the application using normalization principles.
2. **Combine normalized data requirements** from **all user interfaces** into one **consolidated logical** database model.
3. Translate the **conceptual E-R** data model for the application into **normalized data requirements**.
4. **Compare** the consolidated **logical** database design with the **translated E-R** model and produce **one final logical** database model for the application.

### Simple Example of Logical Data Modeling — Conceptual Data Model and Transformed Relations



#### Relations:

```

CUSTOMER(Customer_ID,Name,Address)
PRODUCT(Product_ID,Description)
ORDER(Order_Number,Customer_ID,Order_Date)
LINE ITEM(Order_Number,Product_ID,Order_Quantity)
INVOICE(Invoice_Number,Order_Number)
SHIPMENT(Invoice_Number,Product_ID,Ship_Quantity)
    
```

# Process of Database Design

- Physical Design
  - Based upon results of **logical** database design
  - Key decisions
    1. Choosing **storage format** for each **attribute** from the logical database model
    2. **Grouping attributes** from the logical database model into **physical records**
    3. **Arranging related records** in secondary memory (**hard disks** and magnetic tapes) so that records can be **stored, retrieved** and **updated rapidly**
    4. **Selecting media** and **structures** for **storing** data to make access more **efficient**



# Deliverables and Outcomes

- **Logical** database design must **account** for every **data element** on a system **input** or **output**
- **Normalized relations** are the primary **deliverable**
- **Physical** database design **results** in **converting relations** into **files**

# Relational Database Model

- **Data represented** as a set of **related tables** or **relations**
- **Relation**
  - A **named, two-dimensional** table of data. Each **relation** consists of a set of **named columns** and an arbitrary **number** of unnamed **rows**
  - **Properties**
    - **Entries** in cells are **simple**
    - Entries in **columns** are from the **same** set of **values**
    - Each **row** is **unique**
    - The **sequence** of **columns** can be **interchanged** without changing the meaning or use of the relation
    - The **rows** may be **interchanged** or stored in any sequence

# Designing Forms and Reports

- System **inputs** and **outputs** are produced at the **end** of the **analysis** phase
  - **Precise appearance** was **not defined** during this phase
- **Forms** and **reports** are integrally related to **DFD (Entity Relationship Diagram) (Data Flow Diagram)** and **E-R** diagrams

*DFD and E-R diagram is a visual representation of information flows within your application. It shows how information enters and leaves the application, what changes the information and where information is stored.*

# Designing Forms and Reports:

## Key Concepts

- **Form**

- A business document that **contains** some **predefined** data and may include some areas where additional data are to be **filled** in
- An **instance** of a **form** is typically based on **one** database **record**

- **Report**

- A business document that contains only **predefined** data
- A **passive** document for **reading** or **viewing** data
- Typically contains data from **many** database **records** or transactions

**vs**

# The Process of Designing Forms and Reports

- **User-focused** activity
- Follows a **prototyping** approach
- **Requirements** determination
  - **Who** will **use** the form or report?
  - **What** is the **purpose** of the form or report?
  - **When** is the **report** needed or used?
  - **Where** does the form or report need to be **delivered** and used?
  - **How many** people need to **use** or **view** the form or report?

# The Process of Designing Forms and Reports

- **Prototyping**
  - **Initial** prototype is designed from **requirements**
  - **Users review** prototype design and either **accept** the design or **request** changes
  - If **changes** are **requested**, the **construction-evaluation-request** cycle is repeated until the design is **accepted**

Customer Details

**Elizabeth Andersen**

Go to  E-mail Customer Create Outlook Contact Save and New Close

**General** Orders

**Company**

**Primary Contact**

First Name   
Last Name   
Job Title

**Phone Numbers**

Business Phone   
Mobile Phone   
Fax Number

**Address**

Street   
City   
State/Province   
Zip/Postal Code   
Country/Region

**E-mail**   
**Web Page**

**Notes**

Record: 14 1 of 29 No Filter Search

# Deliverables and Outcomes

- **Design specifications** are **major deliverable** and contain **three** sections
  1. Narrative
  2. Screen Design
  3. Testing and usability assessment



# General **Formatting** Guidelines for Forms and Reports

## • **Highlighting**

- Use **sparingly/ carefully** to draw user **to** or **away** from **certain information**
- **Blinking** and **audible** tones should only be used to highlight **critical** information requiring user's **immediate** attention
- Methods should be **consistently selected** and used based upon level of **importance** of emphasized information

## Methods of Highlighting

Blinking and audible tones

Color differences

Intensity differences

Size differences

Font differences

Reverse video

Boxing

Underlining

All capital letters

Offsetting the position of nonstandard information

# General Formatting Guidelines for Forms and Reports

## Guidelines for Displaying Text

Case	Display text in mixed upper- and lowercase and use conventional punctuation.
Spacing	Use double spacing if space permits. If not, place a blank line between paragraphs.
Justification	Left-justify text and leave a ragged right margin.
Hyphenation	Do not hyphenate words between lines.
Abbreviations	Use abbreviations and acronyms only when they are widely understood by users and are significantly shorter than the full text.

# General Formatting Guidelines for Forms and Reports

- Displaying tables and lists
  - Labels
    - All columns and rows should have meaningful labels
    - Labels should be separated from other information by using highlighting
    - Redisplay labels when the data extend beyond a single screen or page

# General Formatting Guidelines for Forms and Reports

- Displaying tables and lists (continued)
  - Formatting columns, rows and text
    - Sort in a meaningful order
    - Place a blank line between every 5 rows in long columns
    - Similar information displayed in multiple columns should be sorted vertically
    - Columns should have at least two spaces between them
    - Allow white space on printed reports for user to write notes
    - Use a single typeface, except for emphasis
    - Use same family of typefaces within and across displays and reports
    - Avoid overly fancy fonts

# General Formatting Guidelines for Forms and Reports

- Displaying tables and lists (continued)
  - Formatting numeric, textual and alphanumeric data
    - Right-justify numeric data and align columns by decimal points or other delimiter
    - Left-justify textual data. Use short line length, usually 30 to 40 characters per line
    - Break long sequences of alphanumeric data into small groups of three to four characters each



## Tabular Report Illustrating Good Report Design

Place meaningful labels on all columns and rows

Alphabetic text is left-justified

Use a meaningful title

Box the table data to improve the appearance of the table

January 10, 2003

Pine Valley Furniture

Salesperson Annual Summary Report, 2002

Page 1 of 2

Region	Salesperson	SSN	Quarterly Actual Sales			
			First	Second	Third	Fourth
Northwest & Mountain						
	Baker	999-99-9999	195,000	146,000	133,000	120,000
	Hawthorne	999-99-9999	220,000	175,000	213,000	198,000
	Hodges	999-99-9999	110,000	95,000	170,000	120,000
Midwest & Mid-Atlantic						
	Franklin	999-99-9999	110,000	120,000	170,000	90,000
	Stephenson <sup>1</sup>	999-99-9999	75,000	66,000	80,000	80,000
	Swenson	999-99-9999	110,000	98,000	100,000	90,000
New England						
	Brightman	999-99-9999	250,000	280,000	260,000	330,000
	Kennedy	999-99-9999	310,000	190,000	270,000	280,000

1. Sales reflect May 1, 2002 – December 31, 2002.

Superscript characters can be used to alert reader of more detailed information

Sort columns in some meaningful order (names are sorted alphabetically within region)

Long sequence of alphanumeric data is grouped into smaller segments

Right-justify all numeric data

Try to fit table onto a single page to help in making comparisons

# Designing Interfaces and Dialogues

- Focus on **how** information is **provided** to and **captured** from users
- **Dialogues** are analogous to a **conversation** between two **people**
- A good **human-computer** interface provides a **uniform** structure for **finding, viewing** and **invoking** the different components of a system



# The Process of Designing Interfaces and Dialogues

- **User-focused** activity
- **Similar to** form and report designing process
- Employs **prototyping** methodology
  - **Collect** information
  - Construct **prototype**
  - Assess **usability**
  - Make **refinements**

# The Process of Designing Interfaces and Dialogues

- **Deliverables**
  - Design Specifications
    - Narrative
    - Sample Design
    - **Testing** and usability assessment

# Designing Interfaces

- **Designing Interfaces**

1. Designing **layout**,
2. Structuring and
3. controlling **data entry field**,
4. providing **feedback**,
5. designing online **help**

# Designing Layouts

- Designing **Layouts**
  - **Standard** formats similar to **paper-based forms** and **reports** should be used
  - **Screen navigation** on data entry screens should be **left-to-right**, **top-to-bottom** as on paper forms

# Designing Layouts

- **Flexibility** and **consistency** are primary design goals
  - **Users** should be able to **move freely** between fields (e.g. **back & Forth**)
  - **Data** should **not** be **permanently saved** until the user explicitly **requests** this
  - Each **key** and **command** should be assigned to **one function**

# Structuring Data Entry

<b>Entry</b>	<b>Never</b> require data that are <b>already online</b> or that can be computed ( <b>date, age, total sale...</b> )
<b>Defaults</b>	Always provide default values when appropriate ( <b>product price</b> )
<b>Units</b>	Make clear the type of data units requested for entry
<b>Captioning</b>	Always place a caption adjacent to fields ( <b>dd/ mm/yyyy</b> )
<b>Format</b>	Provide formatting examples ( <b>decimal place, comma, \$</b> )
<b>Justify</b>	Automatically justify data entries
<b>Help</b>	Provide <b>context-sensitive</b> help when appropriate

# Controlling Data Input

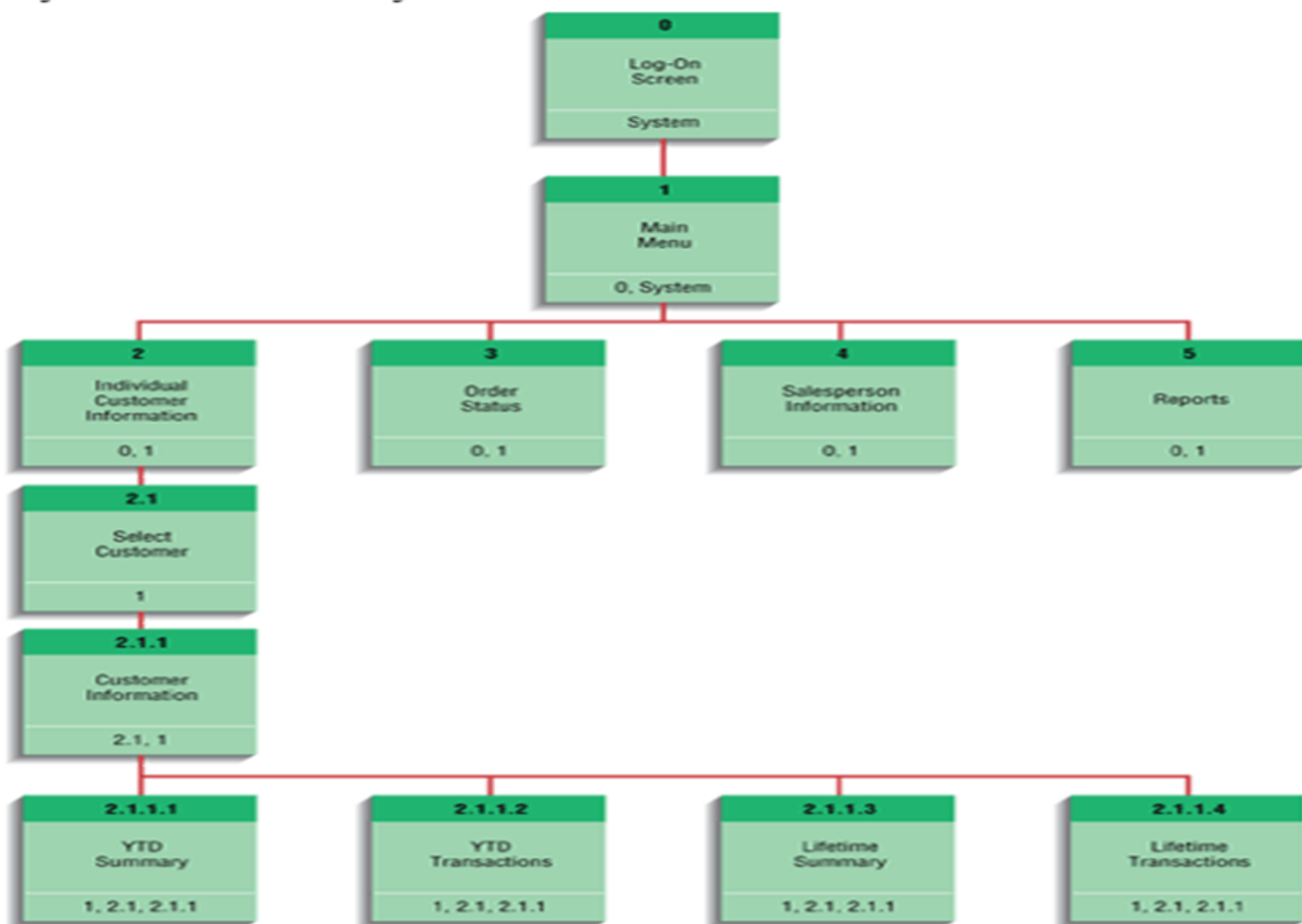
- One **objective** of interface design is to **reduce** data entry **errors**
- Role of systems analyst is to **anticipate** user **errors** and **design features** into the system's interfaces to **avoid**, **detect**, and **correct** data entry mistakes
- **Table 8-9** describes types of data **entry errors**
- **Table 8-10** lists **techniques** used by system designers to detect errors

# Example

- Suppose that the **marketing manager** at Pine Valley Furniture (PVF) wants **sales** and **marketing** personnel to be able to review the **year-to-date transaction activity** for any PVF **customer**
- **The Dialogue sequence**
  - 1. Request to view individual customer information
  - 2. Specify the customer of interest/select customer
  - 3. Select the year-to-date transaction summary display
  - 4. Review customer information
  - 5. Leave system



## Dialogue Diagram for the Customer Information System at Pine Valley Furniture



# Conclusion

- Logical database design is the process of deciding how to arrange the attributes of the entities in a given business environment into database structures, such as the tables of a relational database.
- The goal of logical database design is to create well structured tables that properly reflect the company's business environment.
- Logical Database Design has a low-level description of entities that are defined and how they are related to each other and what kind of data is to be stored.
- This model determines if all the requirements of the business have been gathered.
- **The logical data model represents all user views of a database.**

# Questions and exercises:

1. What is logical design of database ?
2. Who are 4 key steps of logical database design ?
3. What is the part designing Interfaces from logical database design ?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 1. Theoretical basis of Databases

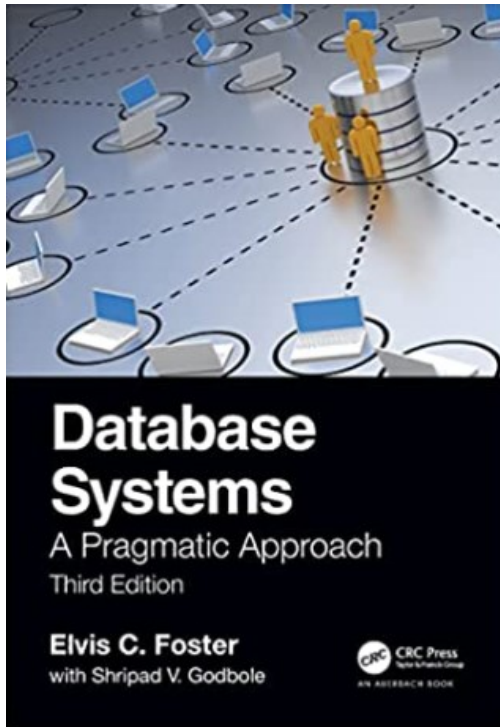
### ❑ Topic 2. Logical and physical bases of databases

#### ❑ Lesson 2. Physical foundations of databases

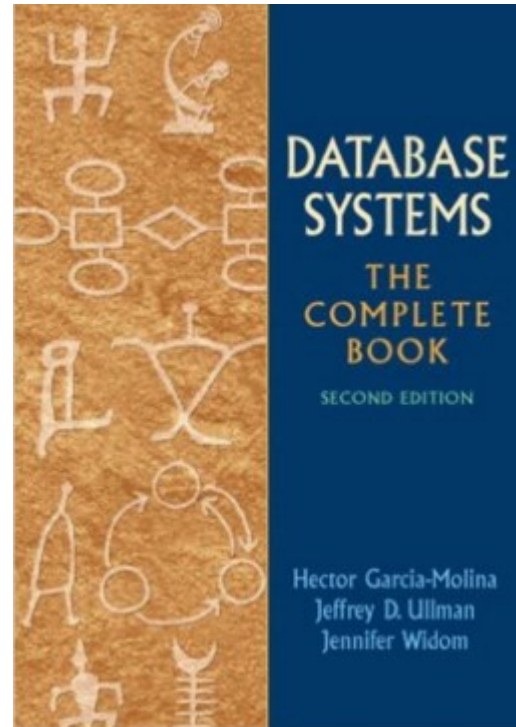


ERASMUS+

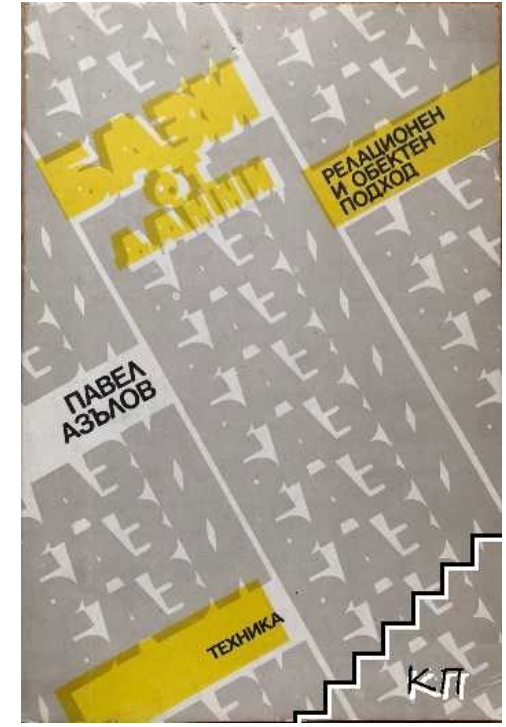
# PHYSICAL FOUNDATIONS OF DATABASES



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

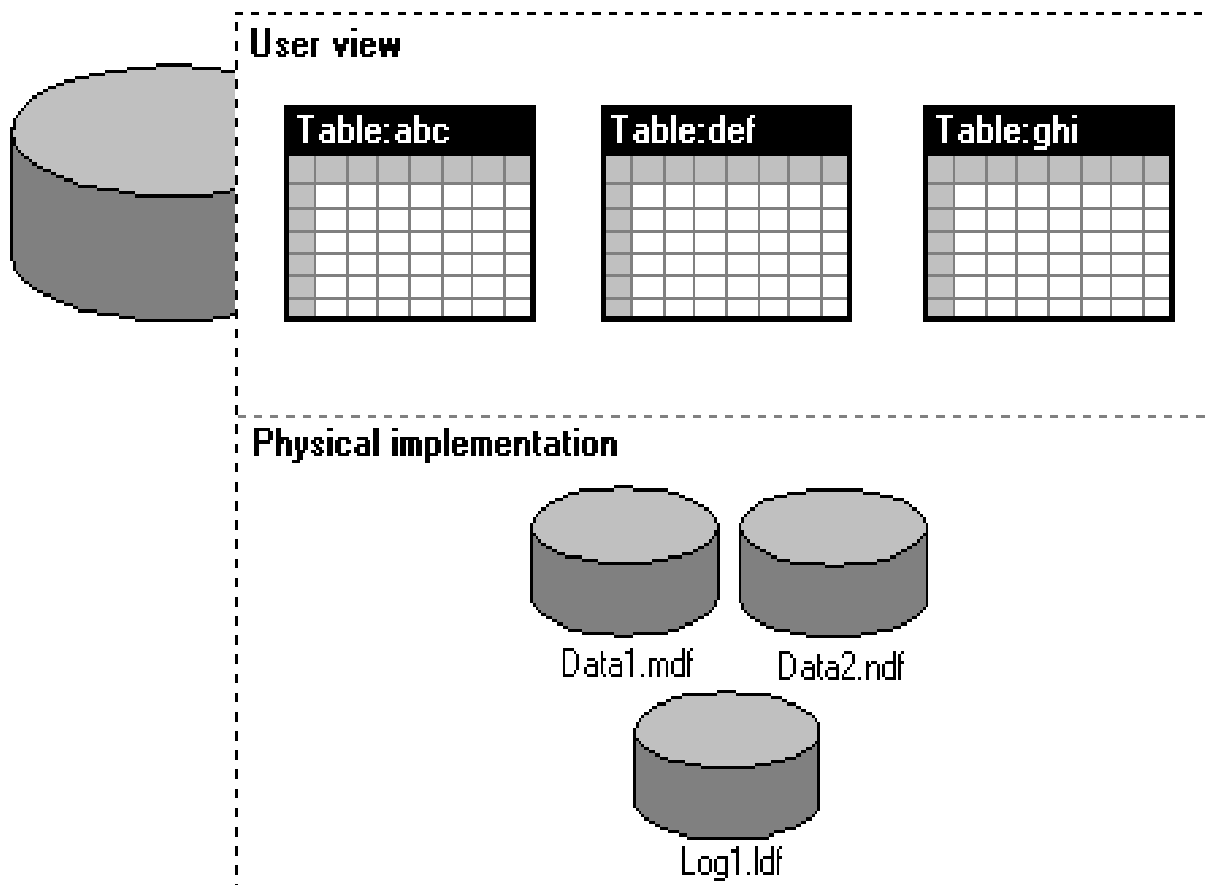
# INTRODUCTION

- The data in a database is organized into logical components visible to users.
- The database is also physically implemented as two or more files on disk.
- When using a database, you work primarily with logical elements such as tables, views, procedures, and users.
- The physical implementation of files is mostly transparent. Usually only the database administrator needs to work with the physical implementation.



# INTRODUCTION

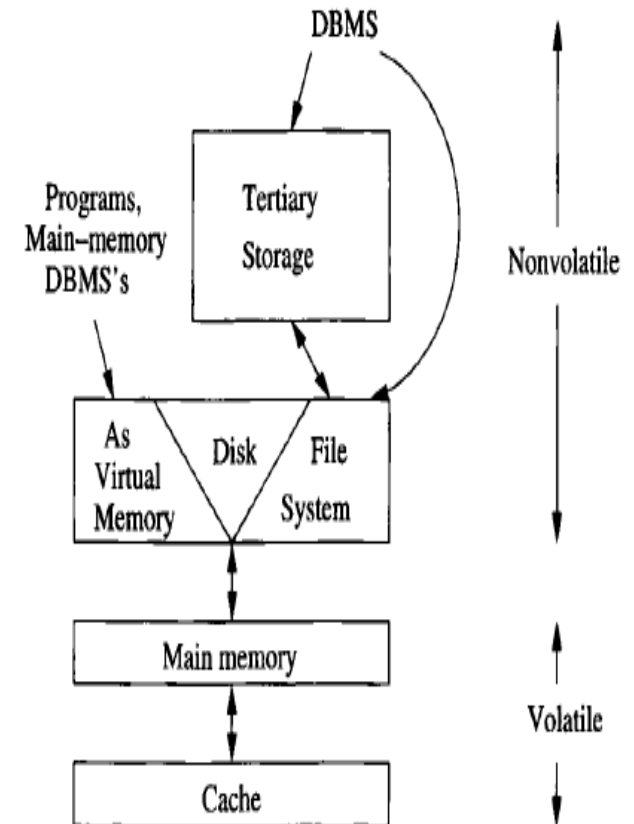
## Database XYZ





# INTRODUCTION

- Typical computer system has several various components where the data can be stored.
- These components have data capacity, varying in different order and also have different access speeds to the data.



The hierarchy of memory

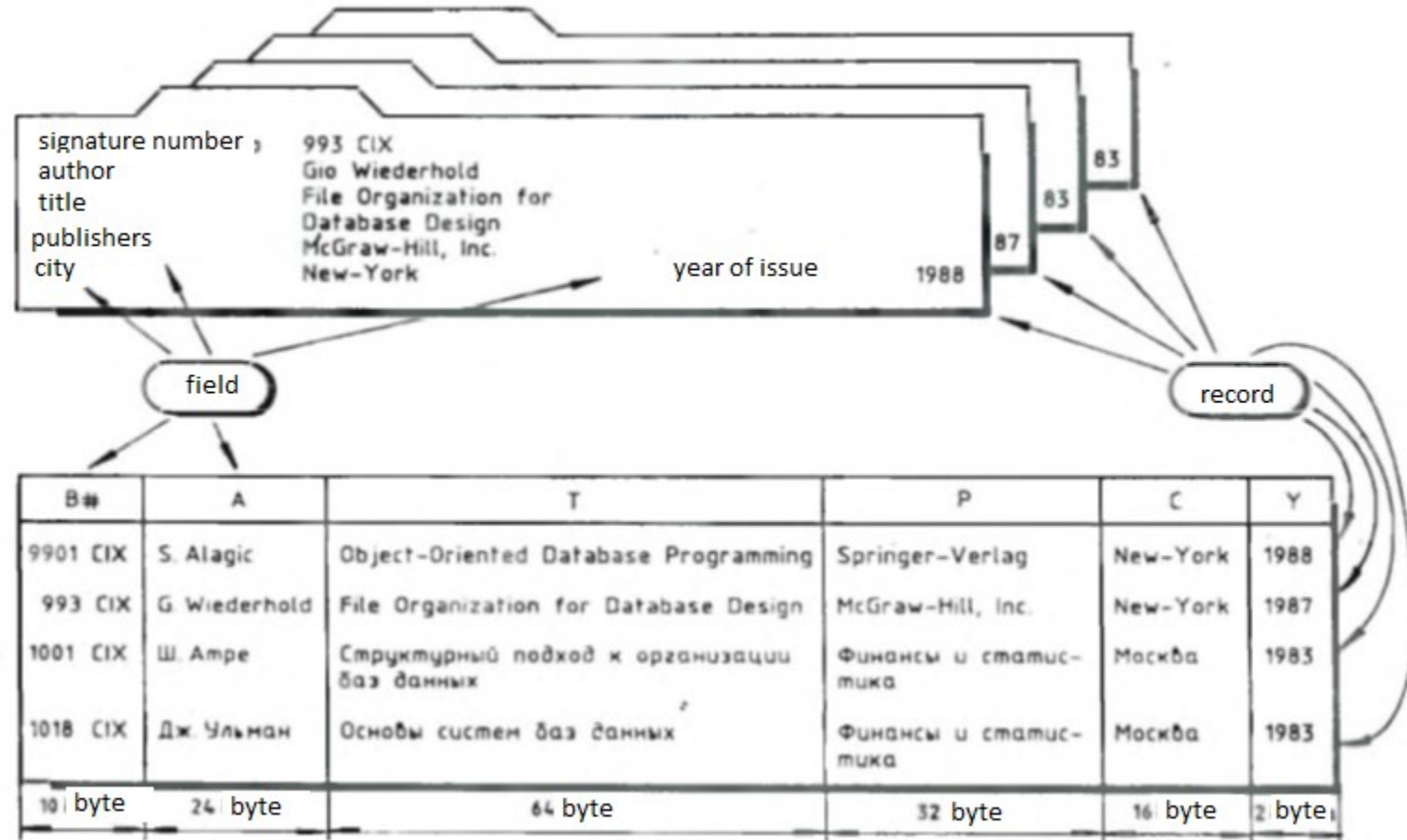
# FILES. BASIC CONCEPTS.

## ➤ RECORDS

- Objects of the same class are described by the same set of attributes.
- Each attribute can receive values from a strictly-defined set of values called the attribute's area.
- The set of values of the individual attributes of a class represents a particular object-sample of that class, often called a record.
- The following figure gives examples of library index-slips, each of which represents a separate copy - a book.

# FILES. BASIC CONCEPTS.

## ➤ RECORDS



# FILES. BASIC CONCEPTS.

## ➤ RECORDS

- The data units (attribute values) that make up the records are also called fields.
- Characteristic of each field is its **length**. In the external representation of the record, this length is measured in number of characters, and in the internal representation - in number of bytes.
- Thus the **total length** of the record is the **sum of the lengths** of the individual fields.
- Most often, the length of the individual records concerning the different copies of the object class is the same. Such records are called fixed-length records.

# FILES. BASIC CONCEPTS.

## ➤ RECORDS

- There are also cases where variable length records are used. This is required in cases where:
  - a) one or more of the fields are with variable length;
  - б) one or a group of fields repeated several times;
  - в) some of the fields are variable.
- Let the summary attribute be added for each book. The content of the corresponding field of each slip will be text whose length cannot be predetermined. Such a field is called a variable length field.

# FILES. BASIC CONCEPTS.

## ➤ RECORDS

- Very often, in order to briefly describe the content of a book or article, keywords are used instead of a summary. The number of keywords referring to individual books usually varies. When it is necessary to characterize the content of the book more precisely, more keywords are used.
- It can be assumed that the keyword attribute has character strings for values, for instance 16 characters long, but for different books the number of keywords will be different.

# FILES. BASIC CONCEPTS.

## ➤ RECORDS

- A field or group of fields (usually of fixed length) that may occur one or more times in the same record is called a **repeating group**.
- Except books, journals are also kept in libraries. Some of the attributes of books and journals are common, but there are also attributes that are different.
- If necessary, books and journals can be considered as objects (copies) of the same class.



# FILES. BASIC CONCEPTS.

## ➤ RECORDS

- This makes entries with two different formats corresponding to books and journals.
- Such records are called variable format records or also variant records.

*Each row of a table is called a record. Records are where individual pieces of information are stored. Each record consists of one or more fields. The fields correspond to the columns in the table.*



# FILES. BASIC CONCEPTS.

## ➤ FILES

- The set of records corresponding to objects of the same class is called a file.
- It is also said that files are used to store on external memory sets of identical data. Each file can be considered as a two-dimensional table.
- The rows in the table represent records and their number determines one of the file dimensions. There are files whose number of records varies from a few records to several hundred thousand records.

# FILES. BASIC CONCEPTS.

## ➤ FILES

- The way in which the individual records are arranged to each other in the file determines the **organization of the file**.
- Sometimes, except the actual data, additional information is stored within or outside the files to make it easier to find individual records.
- The records in the file are searched by specifying certain properties that the desired records must have.
- When formulating these properties, attribute names used in describing the object class are specified.

# FILES. BASIC CONCEPTS.

## ➤ FILES

- It is customary to refer to the attributes by which a file record is searched or sorted as keys.
- A key whose values uniquely identify records in the entire file is a primary key.
- Very often the primary key is defined by a single attribute, but there are cases where it consists of multiple attributes. In such case, the primary key is called a composite key.

# FILES. BASIC CONCEPTS.

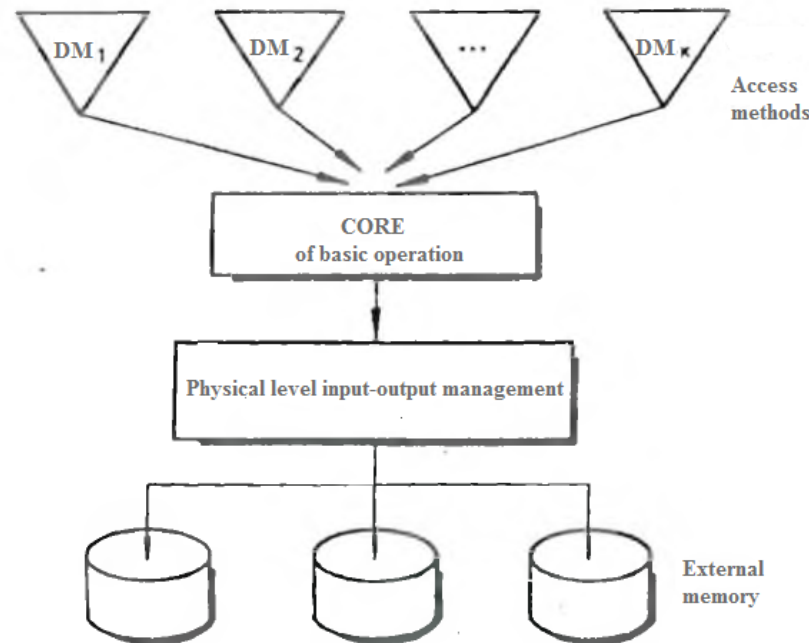
## ➤ FILES

- The way records are searched and selected in the file determines the *access method*.
- There are two main types of access method:
  - consistent access method
  - and*
  - key access method.
- The most used methods of the second type are the methods of access by hashing and by indexing.
- File management is done by a special system called a *file system*.

# FILES. BASIC CONCEPTS.

## ➤ FILES

*Each file system includes a core of basic operations and a set of different access methods:*



**General View of a file system architecture**

# FILES. BASIC CONCEPTS.

## ➤ FILE

- Basic operations are file creation and destruction, disk space allocation, maintaining a catalog for all files in a disk bundle, and managing input-output buffers.
- These operations are used by all access methods that represent an upgrade of the core.
- Each of the access methods interacts with the data from the corresponding files by means of I/O operations to read or write physical records (blocks).

# FILES. BASIC CONCEPTS.

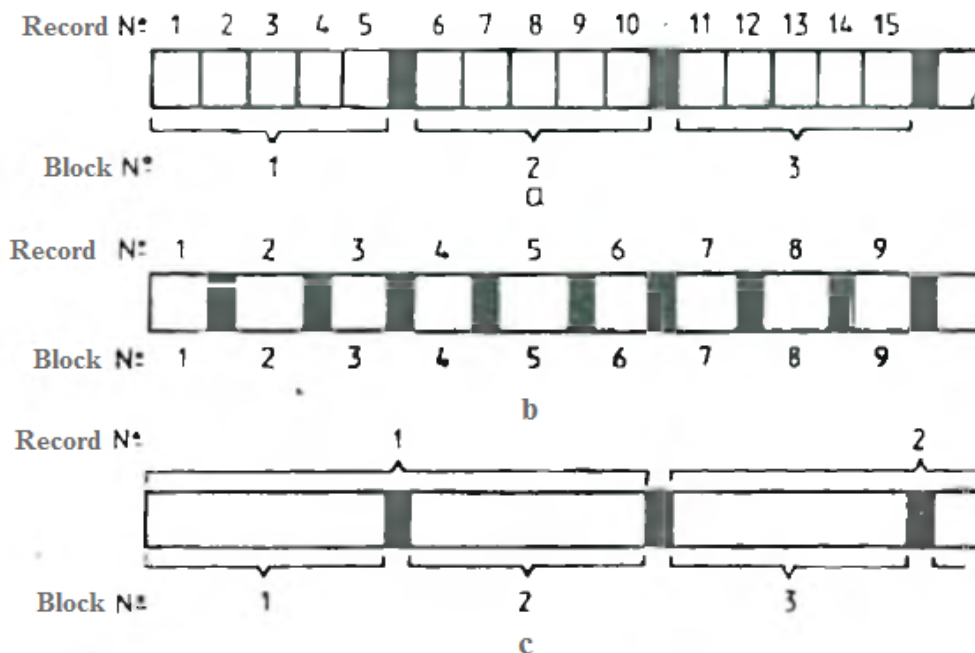
## ➤ FILES

- Usually a block contains several (logical) records and in such case we talk about record blocking.
- However, there are cases where a record is placed on more than one block (see figure on next slide)

*Modern file systems include a variety of access methods.*

# FILES. BASIC CONCEPTS.

## ➤ FILES



Physical and logical records

a – blocking records. Blocking coefficient  $K = 5$ ;

b – no blocking records;

c - one logical record is located on two blocks.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

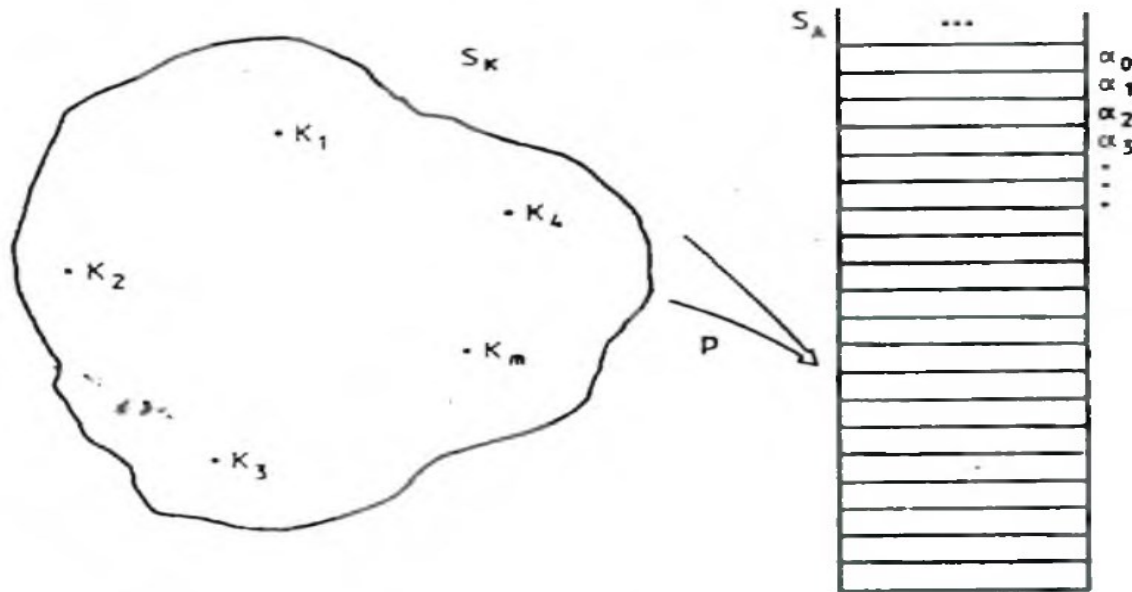
Action Type KA226 - Partnerships for Digital Education Readiness



# PHYSICAL DATA ORGANISATION TASKS.

- A major problem to be solved when creating a concrete physical DB is how the various data structures (tables, trees, grids, etc. ) will be physically represented in sequences of bits.
- As a consequence of this task, several other problems are solved:
  1. *How to determine the location of the corresponding physical record by a given key and possibly in minimum time?*
- In fact, this task leads to establishing a correspondence (rule, procedure, function) between the set of allowable key values for a given SK file (key space) and the set of available addresses from a given external memory area SA.

# PHYSICAL DATA ORGANISATION TASKS.



Correlation between  
the set of classes and  
the set of addresses

- The correlation may be defined directly by a function, in which case it is called a direct access method, and in other cases an additional table called an index may be used, in which case it is called an index access method.

# PHYSICAL DATA ORGANISATION TASKS.

- The other subtasks, which are a consequence of the main task, can be briefly formulated as follows:

*2. How to organize the data so that multi-key search is possible and performed efficiently?*

*3. What should be the physical representation of the hierarchical and network structures and what should be the methods for performing various operations, including the update operation?*

*4. What should be the physical representation of the data where minimum memory is used, i. e. the representation should be of maximum density?*

# PHYSICAL DATA ORGANISATION TASKS.

- Solving some of these tasks optimally leads to less efficient solutions of some of the others.
- This is a consequence of the contradiction that exists in the requirements between the individual tasks.
- Therefore, in the specific design of the physical DB we have to make certain compromises, allowing us to optimally solve the problems of interest in this case.

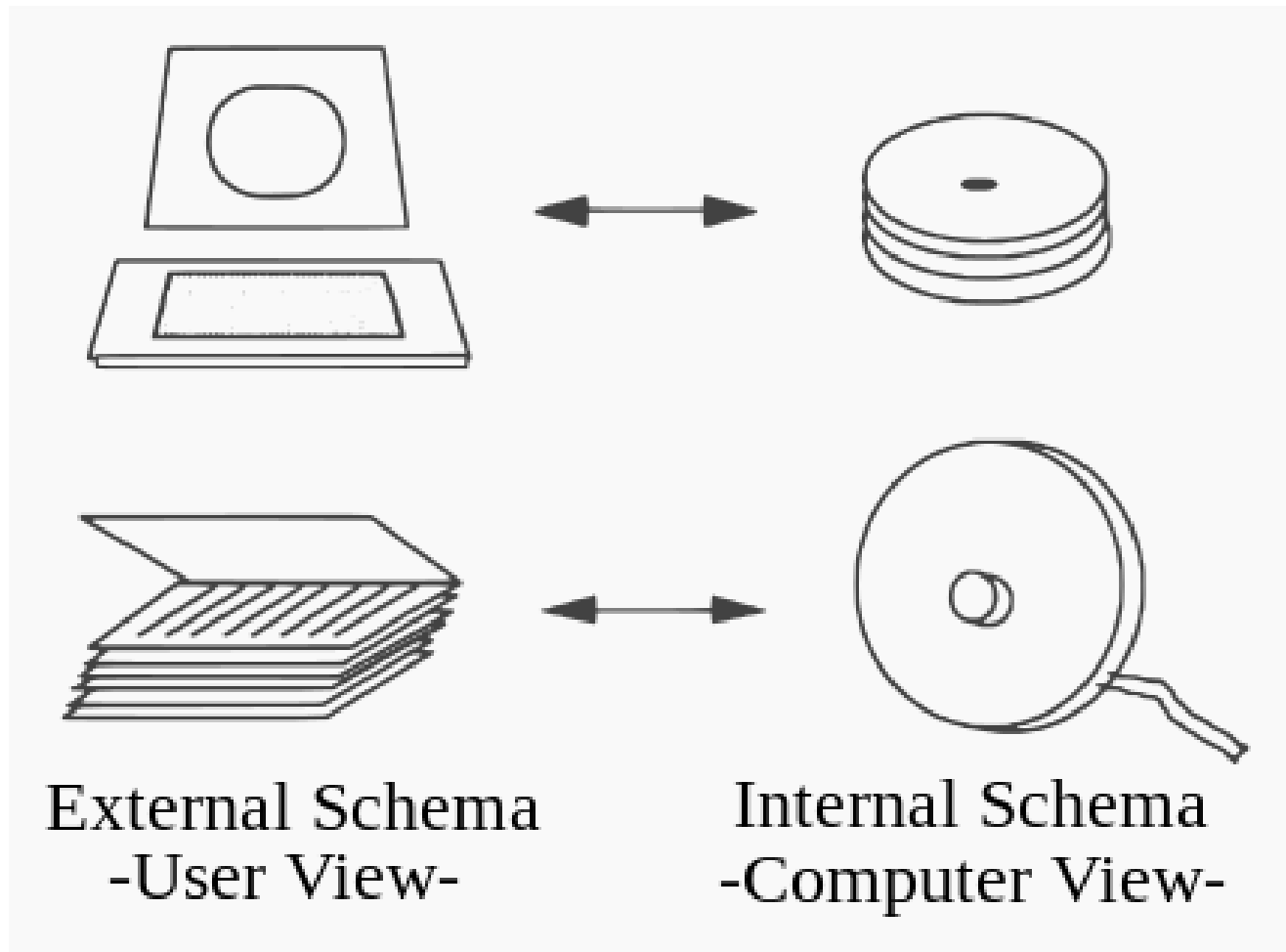
From the basic requirements for the physical organization of data, it is clear that the storage method and the method (algorithm) for accessing the data stored in external memory are the most significant factors that affect the performance of a DBMS.

# PHYSICAL DATA ORGANISATION TASKS.

- Therefore, these characteristics of the internal schema of each DBMS are evaluated with respect to both access efficiency and data storage efficiency.
- *Access efficiency* is defined as a reciprocal value of the average number of I/O operations required to retrieve a logical record from the database.
- *Storage efficiency* is defined as a reciprocal value of the average amount of bytes of external memory required to represent one byte of input data.

# PHYSICAL DATA ORGANISATION TASKS.

- It should be kept in mind here that the physical representation of logical files uses additional tables and control information, reserves more free memory than is needed at the time, etc.
- The most commonly used access methods in modern DBMSs are sequential, index, inverted and hashing access methods, which will be discussed in the next lecture.



## Questions and exercises:

1. What is physical implementation of files in database ?
2. What are the various components where the data can be stored in typical computer system has several ?
3. What is organization of the file ?





# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 1. Theoretical basis of Databases

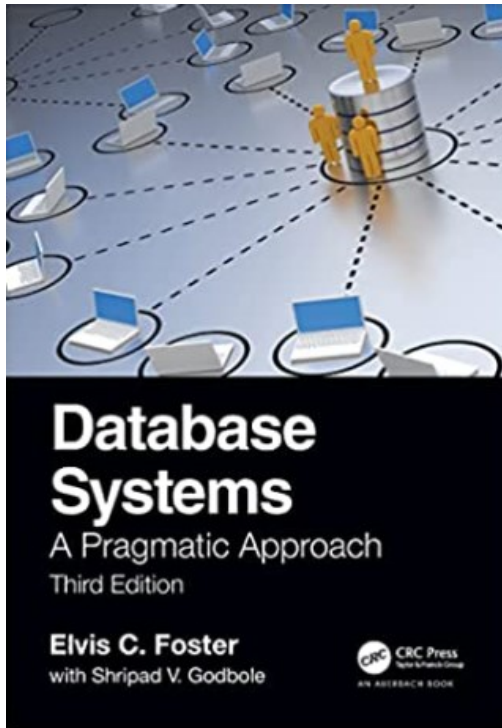
### ❑ Topic 3. Data models

#### ❑ Lesson 1. What is a Database Model. Types of data models.

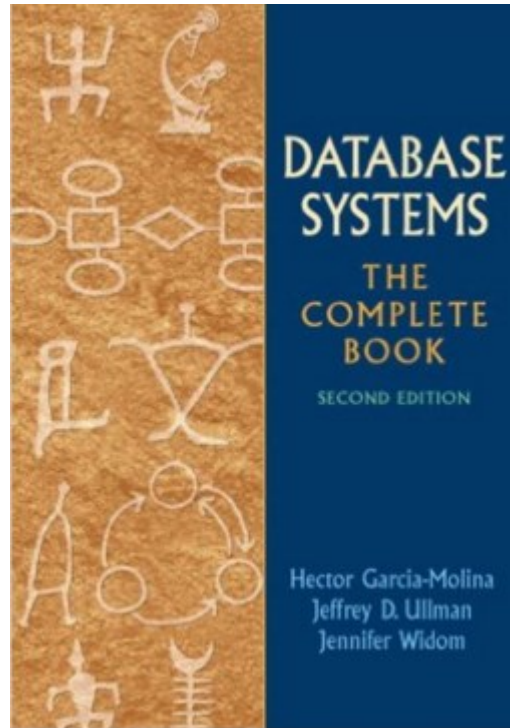


ERASMUS+

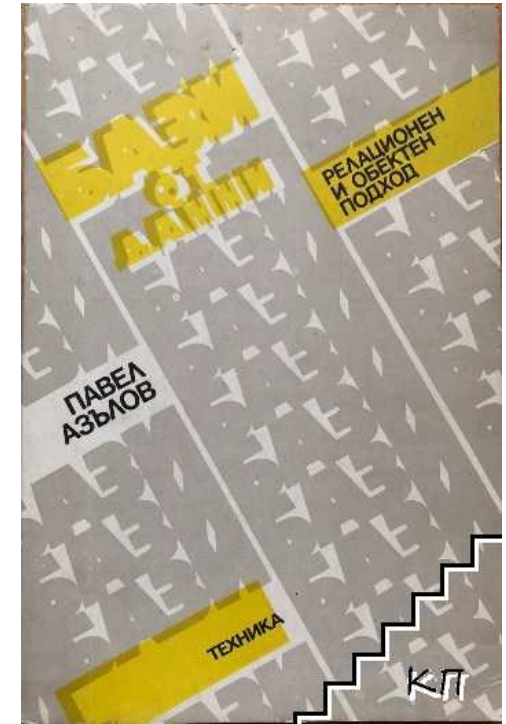
# WHAT IS A DATABASE MODEL. TYPES OF DATA MODELS.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# Introduction

- The idea of a "[data model](#)" is one of the most fundamental in the study of database systems. In this lecture we will define some basic terminology and mention the most important [data models](#).
- Today, mathematical, cybernetic, economic, physical and many other models from different fields of knowledge are known and studied.
- Models are created in the form of drawings, constructions, texts, schemes, equations, algorithms, etc.

# Introduction

- The scheme of each DB is also a model.
- It is a model of a particular subject area, represented by descriptions of a finite number of real-world objects and the relationships that exist between them.
- Data models (DM) have a more specific purpose. They are used for DB schemes, i. e. the means of DM are used to create models of objects for which data are stored and processed by computers.

# Introduction

- Here we will focus on the concept of DM and its relation to the concept of DB.
- The organization of the data in each DB is one of its most important features and it is directly related to the used model.

# What is a data model?

- A data model is a **notation** (notation system) for describing data or information. The description usually consists of three parts:

**1. Data structure.** You may be familiar with tools in programming languages such as C or Java for describing the data structure used by a program: arrays and structures or objects, for example. The data structures used to represent data in a computer are sometimes qualified, when discussing database systems, as a model for the physical data, although in reality they are far away from the way in which the physical data is really represented.



# What is a data model?

*In database world, data models are at a slightly higher level than data structures, and are sometimes referred to as a **conceptual model** to highlight the difference in the level.*

**2. Operations on data.** In programming languages, operations on data are in general something that can be programmed. For data models in databases, there is usually a limited set of operations that can be performed. Usually we can perform a limited set of queries (operations that retrieve information) and modifications (operations that change information in the database). This limitation is not a weakness but a strength. By constraining the operations, it is possible for programmers to describe database operations at a very high level, database management systems to implement operations efficiently are known.

# What is a data model?

**3. Data restrictions.** Data models in databases usually have a way of describing constraints on what the data can be. These constraints can range from simple (e. g. , "a day of the week is an integer between 1 and 7" or "a movie has at most one title") to some very complex constraints.

# Important types of data models

- Today, the two data models are of fundamental importance for database systems:
  1. **The relational model**, including object-relational extensions.
  2. **Semi-structured model** including **XML** and related standards.
- ✓ *The first one (relational model), which is present in all commercial database management systems, is the subject of this course and will be discussed in detail in a separate lecture.*
- ✓ *The semi-structured model, of which XML is the main procedure, is an added feature of most relational DBMSs, and appears in a number of other contexts as well.*

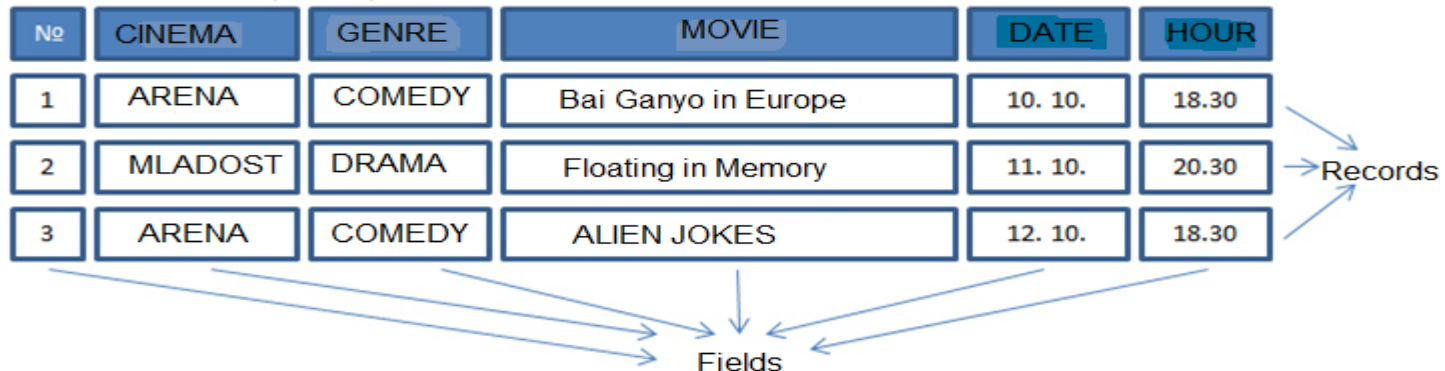
# OBJECTS AND THEIR DESCRIPTION

- Each database consists of **objects** that are linked together. These relationships are depicted in the database by specifying **relations** between objects and their characteristics.
- Data for objects of the same type are placed in **tables**. Each row of the table contains the data for one **object (record)** and each **column (field)** contains the corresponding object characteristics.

No	CINEMA	GENRE	MOVIE	DATE	HOURL
1	ARENA	COMEDY	Bai Ganyo in Europe	10. 10.	18.30
2	MLADOST	DRAMA	Floating in Memory	11. 10.	20.30
3	ARENA	COMEDY	ALIEN JOKES	12. 10.	18.30

→ Records

Fields



# OBJECTS AND THEIR DESCRIPTION

- The term "**object**" is not subject to definition. Objects exist and can be distinguished. Objects are, for example, cars, books, people, houses, projects of residential buildings, etc.
- We also consider intangible objects - events, phenomena, feelings. Each object has properties that characterize it and distinguish it from the others.
- For example, characteristic for each book is its title, author, publisher, year of publication, etc.
- Of course, no matter how detailed we describe an object, it is difficult to give its complete and accurate description.

# OBJECTS AND THEIR DESCRIPTION

- However, if we abstract from the non-essential properties of individual objects, a common set of characteristic properties can be defined for each object that represents it accurately enough.
- In this way, sets of identical (similar) objects can be considered, possessing some common properties.
- Each such set will contain objects that differ from each other in the values of all or some of their characteristic properties.
- Let's consider, for example, the sets of objects "personal cars", assuming that the properties that characterize each car are: license plate number, owner, and year of manufacture.



# OBJECTS AND THEIR DESCRIPTION

- A few examples of objects from this set are given in the table:

Plate number	Owner	Year of manufacture
C 1584	Ivan Petrov	1985
BH5047	Todor Ivanov	1988
C36382	Hristo Ganchev	1979
PA4766	Todor Ivanov	1989

- It is customary to call sets of objects of the same type object classes, and the properties that characterize them - attributes.
- Each attribute is a quantity that is defined by a name and a set of permissible values or, as it is more commonly called, an attribute field.



# OBJECTS AND THEIR DESCRIPTION

- It is permissible for different attributes to have the same field. For example, "Faculty Number" and "Office Number" can be attributes of different classes of objects, but with the same field - the set of positive integers.
- “Student’s name”, “Director’s name”, “Author’s name” are examples of other attributes that also have a common field - the set of alphabetic strings of no more than 40 characters.
- For each object class, **one** or a **group of several attributes** can be defined that straightly identify each object of the class.

# OBJECTS AND THEIR DESCRIPTION

- This attribute or group of attributes is called the object class key.
- Each object class has at least one key. This follows from the fact that object classes contain only objects that are distinct from each other.
- *Examples:*
  1. Class of objects “EMPLOYEE”
    - Class attributes: **office number**; address; date of birth; education; subject; department; payroll.
    - Class key "EMPLOYEE" is the attribute “**office number**”.

# OBJECTS AND THEIR DESCRIPTION

- *Examples:*
- 2. Class of objects “PURCHASE”
  - Class of objects attributes: invoice number; purchase; number; date; buyer.
  - Since more than one item can be purchased with one invoice, it follows that the "invoice number" attribute does not uniquely identify the items in the "PURCHASE" class.
- It can be seen, however, that the first two attributes together uniquely identify the objects in this class.

# OBJECTS AND THEIR DESCRIPTION

- Examples:*

Attribute name	No.	Name	Unit	Marital status	Position	Commence ment of work
	2867	Angel Hristov	05	1	lathe operator	150681
	3704	Stefan Toshev	08	1	caster	280475
	5663	Boris Kostov	05	1	lathe operator	170580
	8457	Kiril Penev	05	0	lathe operator	261182
	9680	Nikola Velev	12	1	millar	190979
Description of one emplooyee						

Values from one  
and the same field

# OBJECTS AND THEIR DESCRIPTION

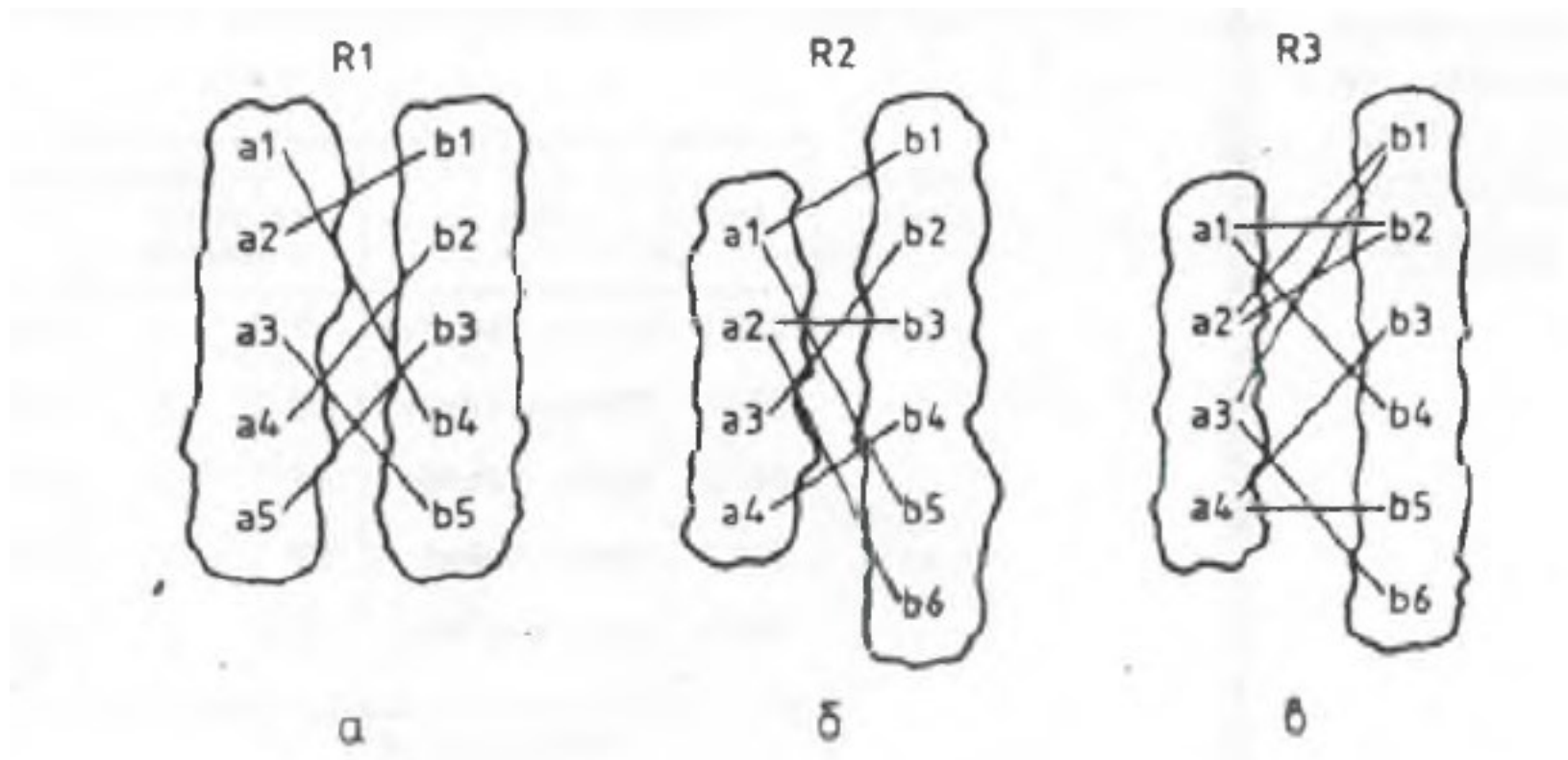
- One of the important stages in the design of any DB modeling a particular subject area is the **selection of the object classes**, their respective sets of attributes, as well as the relationships and relations that exist between the objects of the various classes.
- When creating object models, it is necessary to study the relationships between the attributes describing each class.
- The values that each individual attribute may have make no sense on their own. This can be seen if we consider separately the values C 1584, BH5048, C36382 and ПЛ4766 of the attribute “Plate number” and the values 1985, 1988, 1979 and 1989 of the attribute “Year of manufacture”.

# OBJECTS AND THEIR DESCRIPTION

- However if we learn that a car with plate number C 1584 is produced in 1985, and the cars with plate numbers BH5047, C36382 and ПЛ4766 are produced in 1988, 1979 and 1989 respectively than the relation (correspondence) between these two sets of values give them a sense.
- In this case, to indicate that a relationship has been established between the values of the attributes "Plate number" and "Year of manufacture", the corresponding elements are placed in the same order.
- Let A and B be two sets, and R is a bipartite relation between A and B, i. e.  $R \subseteq A \times B$ . If  $\langle a, b \rangle \in R$ , then **a** is also said to be in relation R with **b**.

# OBJECTS AND THEIR DESCRIPTION

- The figure below shows three examples of relations in which the pairs of elements belonging to the relation are joined by a line.





# OBJECTS AND THEIR DESCRIPTION

- In a given relation  $R$ ,  $R \subseteq A \times B$ , we're considering the relation  $R^{-1}$  defined in the following way:

$$\langle b, a \rangle \in R^{-1} \iff \langle a, b \rangle \in R$$

- The relation  $R^{-1}$  is called reverse on relation  $R$ .
- A special class of relations are the so-called functional relations (or just function) if:

$$\langle a, b \rangle \in R \quad \text{u} \quad \langle a, c \rangle \in R \implies b = c$$

- This property expresses the fact that each element  $a \in A$  is in a relation  $R$  with one and only one element of  $B$ .

# OBJECTS AND THEIR DESCRIPTION

- Consider the examples in the figure above. According to the definition, the relation  $R1$  is a functional relation:

$$R1 = \{ \langle a1, b4 \rangle, \langle a2, b1 \rangle, \langle a3, b5 \rangle, \langle a4, b2 \rangle, \langle a5, b3 \rangle \}$$

- The inverse  $R1^{-1}$  of relation  $R1$  is also a functional relation.
- The  $R2$  relation is not functional because:

$$\begin{aligned} \langle a1, b1 \rangle &\in R2 \\ \langle a1, b5 \rangle &\in R2 \quad \vee \quad b1 \neq b5 \end{aligned}$$

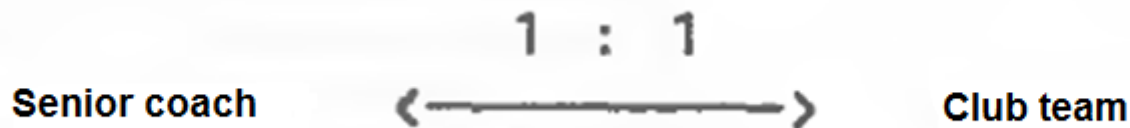
# OBJECTS AND THEIR DESCRIPTION

- Using relations defined over two sets of objects, can be established relations of the form **1:1**, **1:N** or **M:N**.
- *A relation is of the form 1:1 if it is established by a relation  $R$ , which, like its inverse relation  $R^{-1}$ , is functional.*
- *If  $R$  is not functional, and  $R^{-1}$  is functional, than the relationship type is **1:N**.*
- *And finally, if  $R$  and  $R^{-1}$  are not functional, the relationship type is **M:N**.*
- A few examples of different types of relation follow:

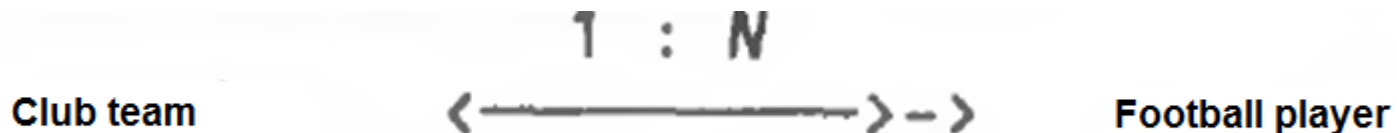
# OBJECTS AND THEIR DESCRIPTION

- Examples:*

1. Each club football team has a single senior coach.



2. Each football player is member of only one football team, but one team consists of many football players.



# OBJECTS AND THEIR DESCRIPTION

- Examples:*

3. Except in a club team a football player may play in the national team as well. Thus each football player is matched one or two teams, and on each team – many of players.



- When an element of one set can be matched to 0, 1 or more elements of the other set, then this is graphically represented by a double arrow. Such are the cases of examples 2 and 3

# OBJECTS AND THEIR DESCRIPTION

- The representation of relationships between object classes is specific to each DM.
- While the relationships between attributes are typically represented in the considered manner, there is more variety in the representation of relationships between objects of individual classes.
- The most popular approaches are those in which relationships between classes of objects are represented by relations or graphs (networks).
- They are the basis of the most common models - relational and graph DMs.

## Questions and exercises:

1. What is a data model in a DBMS and what are the main types of data models ?
2. A data model is a notation (notation system) for describing data or information. The description usually consists form ? ...
3. Explain the terms object in DBMS.





# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 1. Theoretical basis of Databases

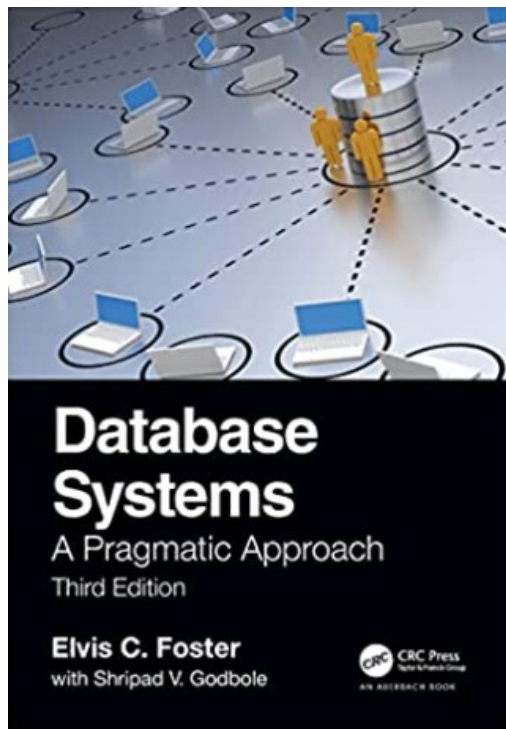
### ❑ Topic 3. Data models

❑ Lesson 2. Database Languages in DBMS: Data description languages. Data manipulation languages.

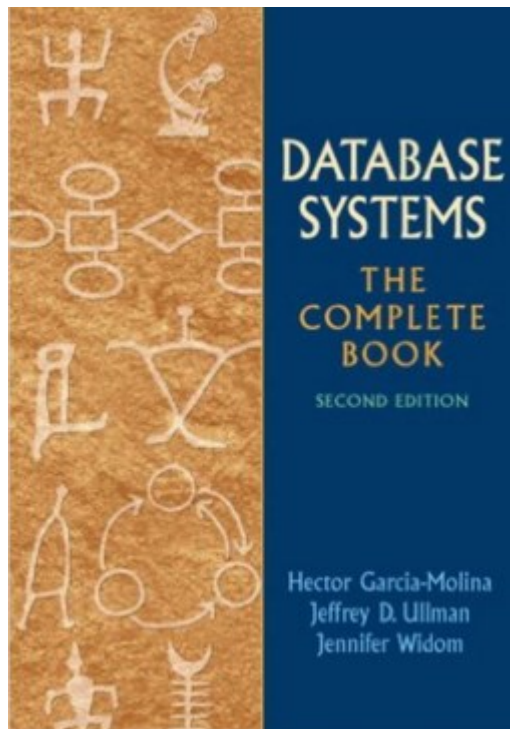


ERASMUS+

# Database Languages in DBMS: Data description languages. Data manipulation languages.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# Data definition language

- Each DB stores data about one or several object classes, as well as data representing the existing relationships between them.
- Each object class is represented by a finite number of attributes.
- Most often the data for the object classes is separated into separate logical files.
- This means that each class of objects is represented by one type of logical record, and the descriptions of the individual objects of each class are instances of logical records:

# Data definition language

Logical file "Private car"

Attributes

Registration number	Owner	Year of manufacture
C 1584	Ivan Petrov	1985
BH5047	Todor Ivanov	1988
C36382	Hristo Ganchev	1979
Π4766	Todor Ivanov	1989

Class of objects  
"Private car"



Logical record

# Data definition language

- To ensure a high degree of logical and physical independence of the data, three levels of data description are considered, respectively called **external schema**, **conceptual schema** and **internal schema**.
- **The external schema** is a description of a part of the DB contained in a given user program and represents the particular user's view of the DB structure. It also defines the perimeter of action of the specific user.



# Data definition language

- DB data is stored in external memory most often on magnetic disks and tapes and this requires that storage structures such as files, records, fields, indexes, hashing methods, etc. to be described. This description, oriented towards the physical storage of the data itself, is the internal schema.
- **The internal schema** is a description of the physical parameters of the DB data, which includes a specification of the storage structures and access methods for the data stored in external memory. Each DB has a single internal and one or more external schemes.



# Data definition language

- **A conceptual schema** is a global logical description corresponding to the classes of objects and their relationships in the subject area under consideration.
- Each class and each relationship is represented by a corresponding set of attributes.
- It may be further added that the conceptual scheme is a description of the subject area under consideration, expressed in terms of the relevant DM.
- **For example, the conceptual schema in a relational model is a set of descriptions of tables and logical constraints on the data contained in those tables.**

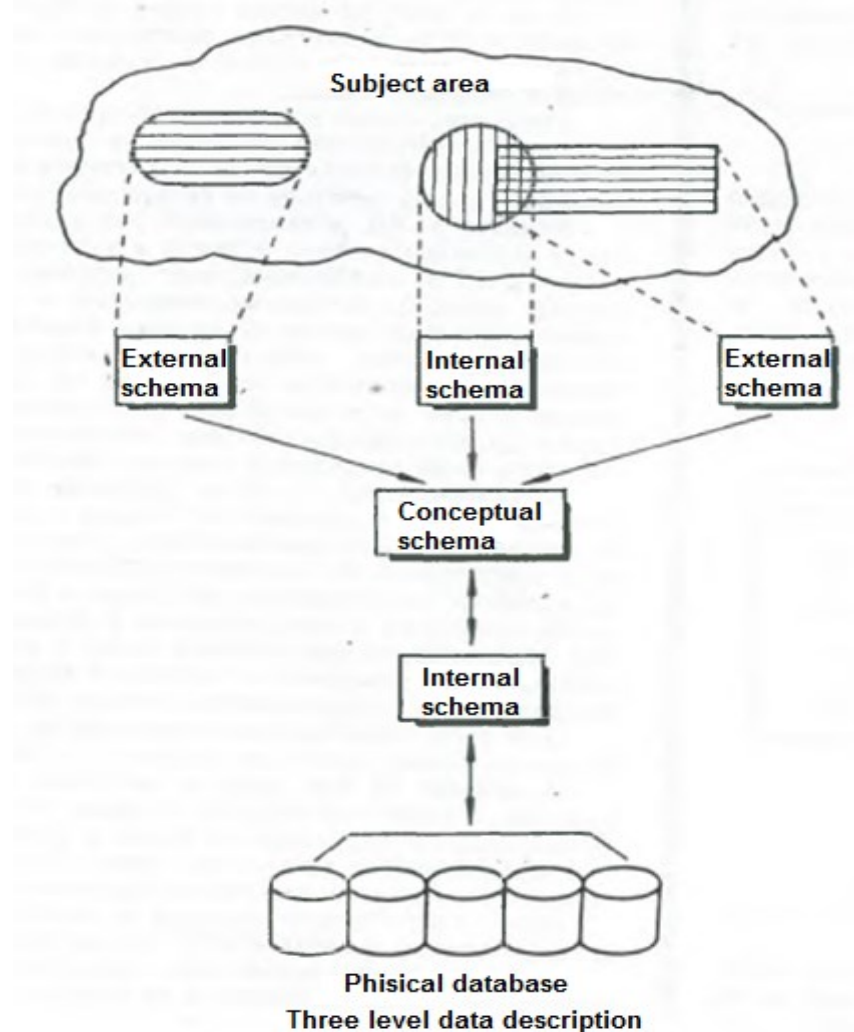
# Data definition language

- Having three levels of data description requires a means by which data structures from one level can be displayed to data structures in another level.
- These tools are algorithms that are implemented with procedures and functions and are called data representations.
- In addition to the structural properties of data, **data definition languages (DDLs)** also define certain dependencies and constraints that may exist between attributes.
- **For example**, for the attribute AE (age of employee) the following condition can be defined:  $17 \leq BC \leq 61$

# Data definition language

- It is customary to refer to such logical conditions on the values of various attributes defined in the DB schema as integrity constraints.
- In general, integrity constraints are a set of rules for maintaining non-contradiction in the data contained in a DB.
- Integrity constraints impose constraints on both the data values and the relationships that exist between them.

# Data definition language



ERASMUS+

# Data manipulation languages

- Very often, the operations defined in an MD are referred to as a **data processing language**. The term **data manipulation language (DML)** has also been used in some literature.
- DML can be an extension of a particular programming language, such as PL/1, PASCAL, FORTRAN, ASSEMBLER, in which case it is called a data sublanguage.
- There are also quite a few DMLs that are completely self-contained and independent of other languages. Such are, for example, the so-called query languages

# Data manipulation languages

Generic DDL Statements	
<i>Statement</i>	<i>Explanation</i>
Create-Database	Creates a database. An Oracle database is a complex object that stores various other database objects and resources.
Alter-Database	Modifies the database.
Drop-Database	Removes the database from the system catalog.
Create-Tablespace:	Creates a tablespace. A tablespace is a logical container within a database. It can contain several datafiles that contain the actual database objects.
Alter-Tablespace	Modifies a tablespace
Drop-Tablespace	Removes a tablespace from the system catalog.
Create-Table	Creates a database table. The table may be a relational table, an object table, or an XML table; the default is relational.
Alter-Table	Alters the physical and/or logical structure of a table.
Drop-Table	Removes a table from the system catalog. The table and all its data are deleted. Due to referential integrity, the typical DBMS forbids you to delete a table that contains referenced tuples.

## Commonly Used DDL Statements

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# Data manipulation languages

- Typically, the scope of the DML operations is some part of the DB, and this is achieved by data selection.
- Thus, in this case, the DML operations include a selection of data and actions permissible over that data
- For example with the operation:

```
select EMPLOYEE - NAME , EMPLOYEE - NUMBER  
from PERSONNEL  
where CITY = ' VARNA '
```

- the names and numbers of the employees from the city of Varna, whose information is stored in the file PERSONNEL will be chosen.



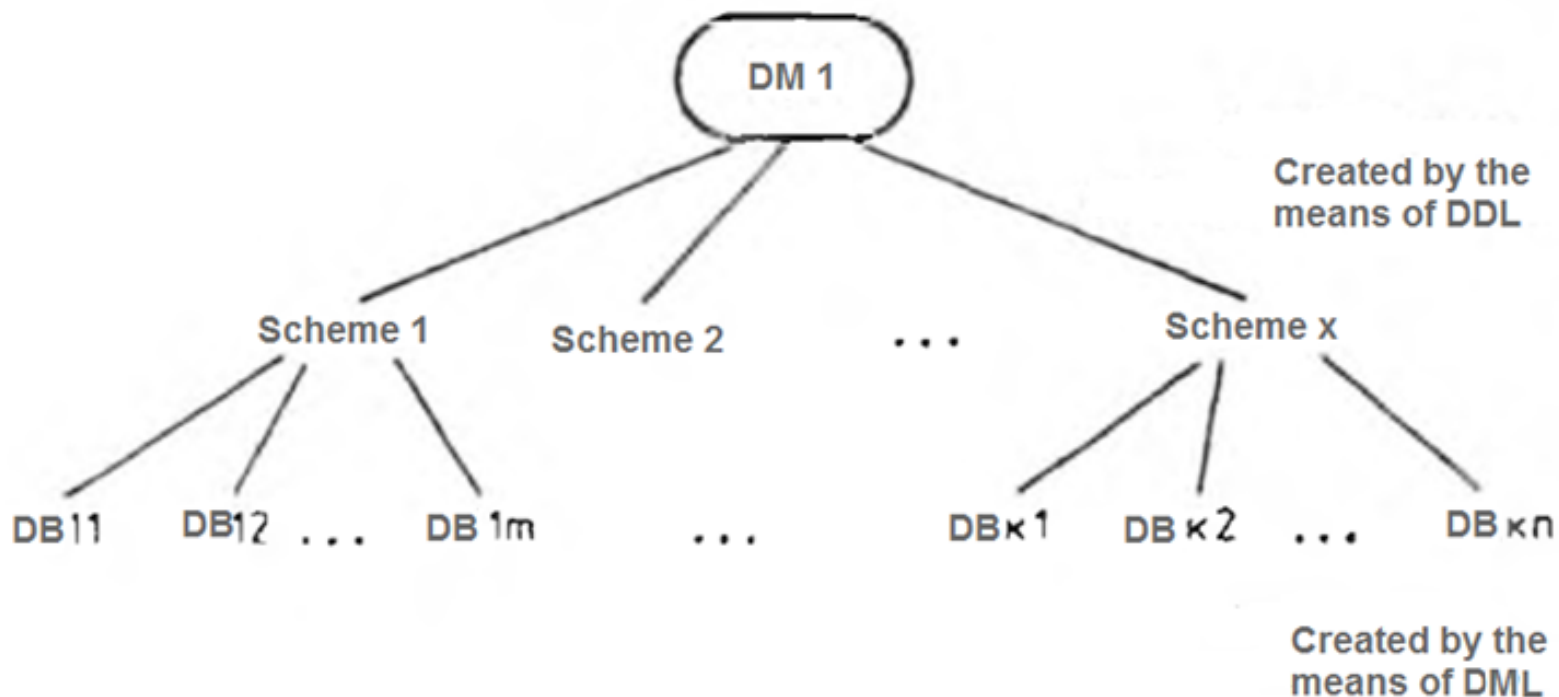
# Data manipulation languages

- The main actions with the DB data are of two types. With one of them, the actual processing of the data is performed, such as:
  - *calculating values of expressions (arguments in them are numbers from DB)*
  - *finding the maximum, minimum, average or sum of numbers selected from the DB.*
- The other type of action makes changes to the state of the DB. Such actions are:
  - Adding new data in DB;
  - Removing data from DB;
  - Changing data in DB.

# Data manipulation languages

- Any operation containing actions of the second type brings the DB into a new state, which, like its old state, must correspond to its schema.
- This means that one of the requirements for the operations of the DML is that they do not destroy the structural properties of the data defined by the DML, i. e. that the data stored in the DB correspond to the requirements given in the schema
- It becomes clear that each DM provides by the means of DML the possibility to generate a multitude of schemes:

# Data manipulation and data definition languages



**Generating multiple schemes of the same model**

# Data manipulation and data definition languages

- From each such schema, a DB is generated which, using DML operations, can modify its contents within the integrity constraints described in the schema.
- Thus, with the data model DM1, the Scheme 1 is generated.
- The operations from DML created the database DB11, which later changed its state and has constantly obtained DB12,. . . , DB1m.

# Graph data models

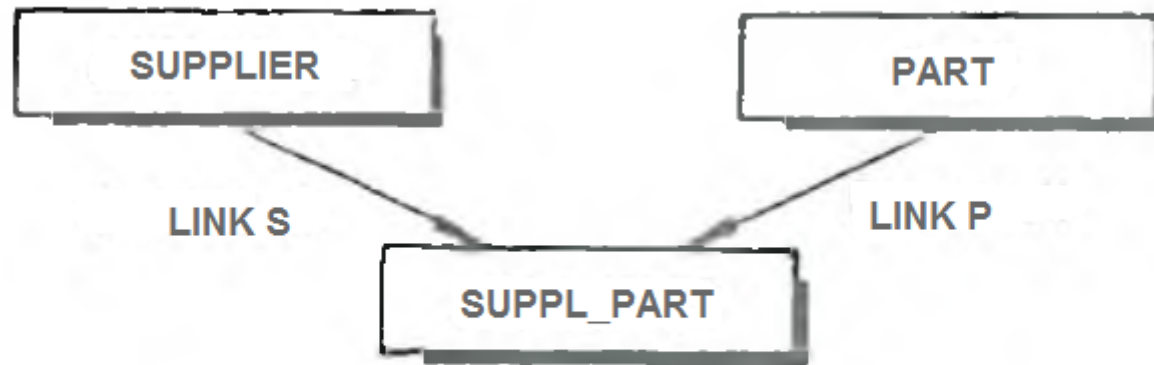
*The concept of the data definition language and its name related to the database model, where the schema of the database was written in a language syntax describing the records, fields, and sets of the user data model. DDL can be generated from a data model.*

- Graph data models were first introduced and used in database practice.
- These are models where the relationships between classes of objects are represented by graphs.
- Typical representatives of graph models are **network** and **hierarchical** data models.
- To the graph data models also refers the data model "**Entity - Relationship**". In the literature it is known as the ER data model, which stands for Entity Relationship.

# NETWORK DATA MODELS

- In general, network models offer the possibility of representing data and the connections between them by a graph, the vertices of which are record types and the arcs of which are link types.
- The graphical representation of a graph is also called a data structure chart. The figure below shows the structure diagram of the “SUPPLIES” database. It has three types of records and two types of links.

# NETWORK DATA MODELS



**STRUCTURAL CHART OF "SUPPLIES" DATABASE**

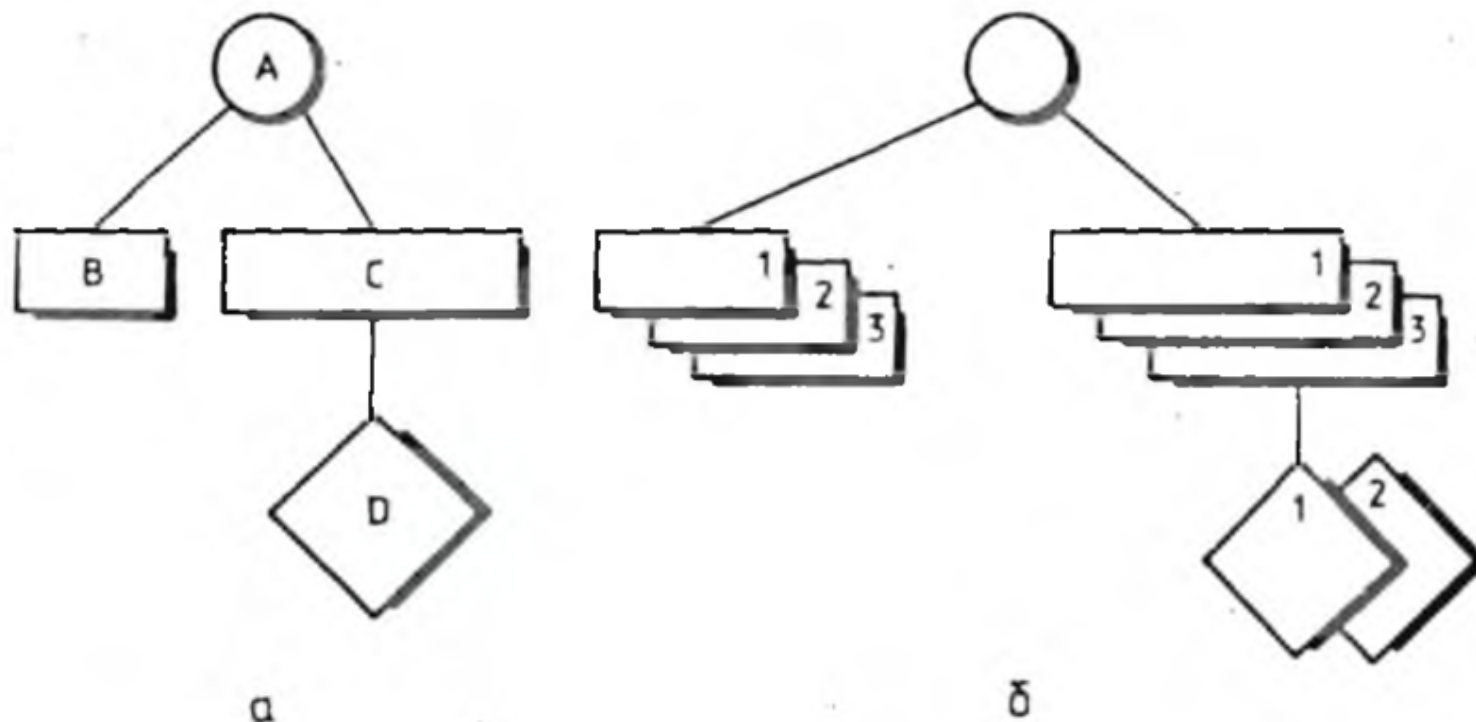
Let  $R1$  and  $R2$  be two record types. The  $L12$  link type between  $R1$  and  $R2$  sets the correspondence between the elements (records) of  $R1$  and  $R2$ . This means that for each record  $r1$  of  $R1$ , a set of records from  $R2$  corresponding to  $r1$  can be determined by the  $L12$  link.



# HIERARCHICAL DATA MODELS

- Both network and hierarchical data models are graph models. In hierarchical models, however, stronger constraints are imposed on the relationships between classes of objects:
  1. Links are functional and are not named.
  2. The structure diagram of a DB is well ordered tree, i. e. a tree in which the order of the individual subtrees is relevant.
- Each node in a structure diagram, also called a definition tree, is a record type.
- The hierarchical database is a collection of samples of the definition tree.

# HIERARCHICAL DATA MODELS

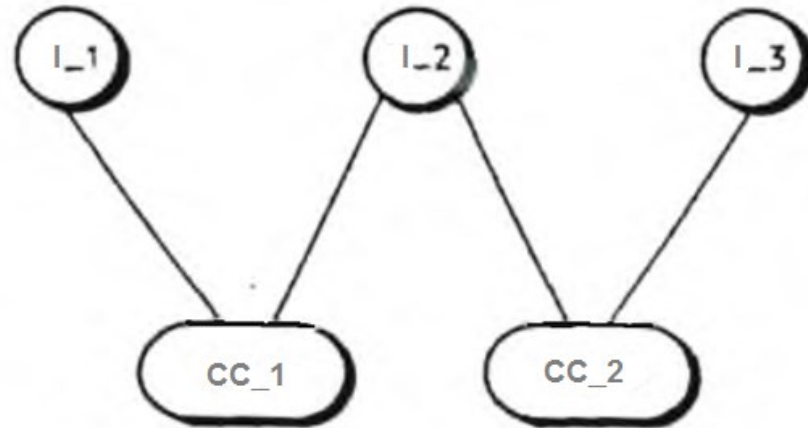
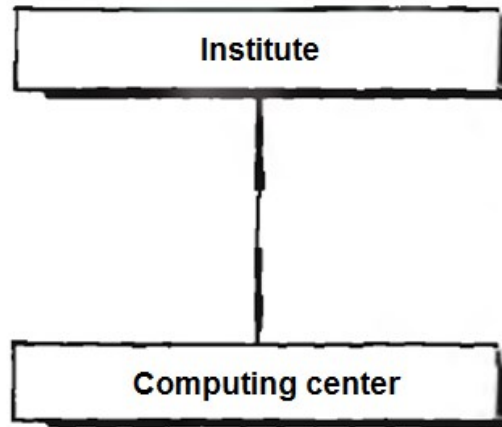


Hierarchical database  
 $\alpha$  - Definition tree  
 $\delta$  - A sample of the definition tree

# HIERARCHICAL DATA MODELS

- *How network structure diagrams are represented by trees?*
- The structure diagram in the figure below expresses the fact that one institute can be served by the computing facilities of several computing centers, and one computing center can serve several institutes, i. e., a relationship of the form M:N is established between the object classes Institute (I) and Computing Center (CC).
- Since no such relationship is supported in the hierarchical model, the following approach can be taken:

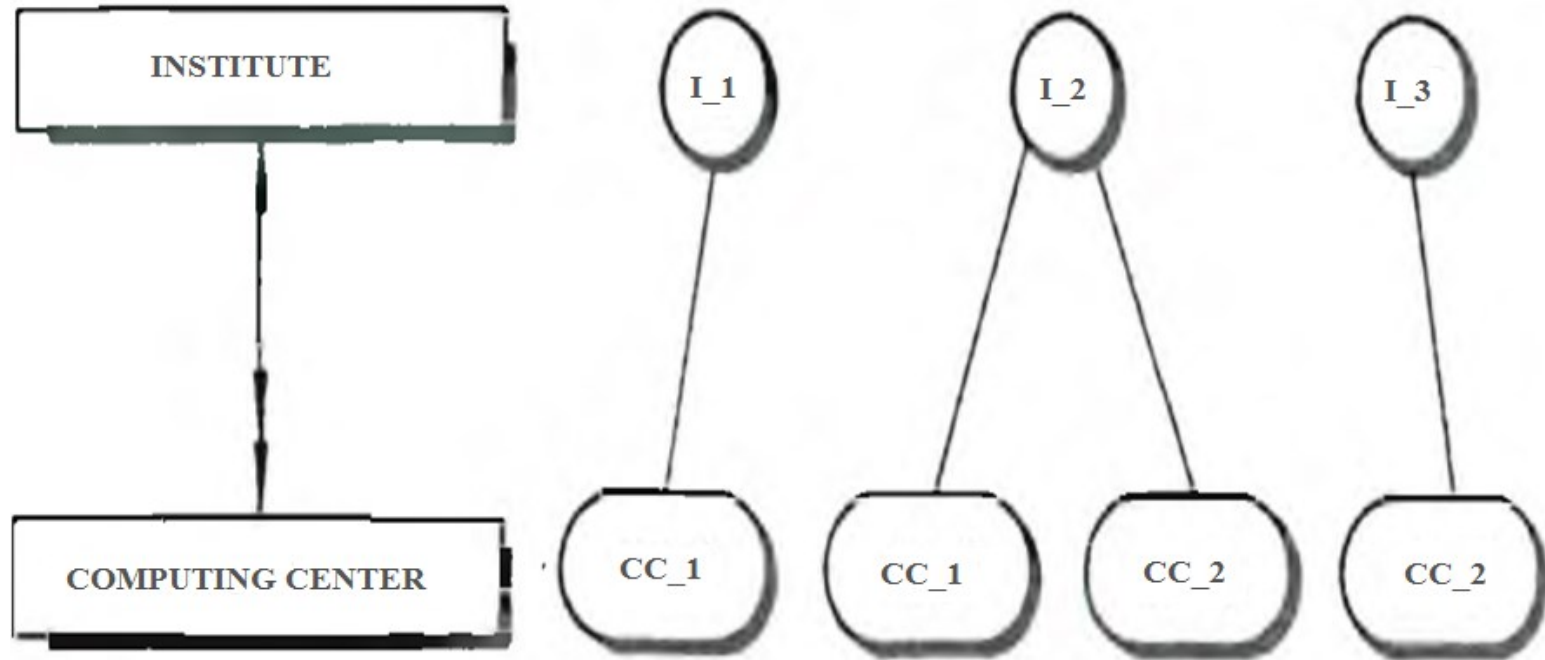
# HIERARCHICAL DATA MODELS



*Structure diagrams two classes of objects*

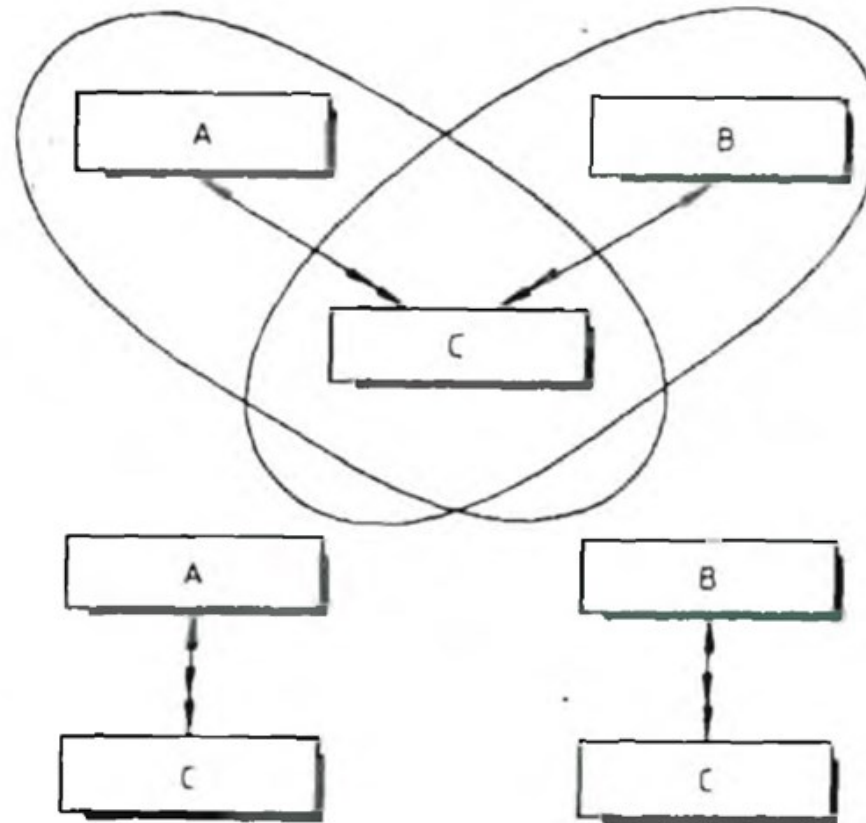
- One of the classes is assumed to be the basic and the other is assumed to be the subordinate. Under the assumptions of the figure given below, the fundamental class is "Institute" and a question like:
- Which computing centers serve the I-2 institute? can be answered easily. However, it is more difficult to answer the opposite question:
  - Which institutes are served by the CC-2 computing center?

# HIERARCHICAL DATA MODELS



**Definition tree with a basic class "Institute"**

# HIERARCHICAL DATA MODELS



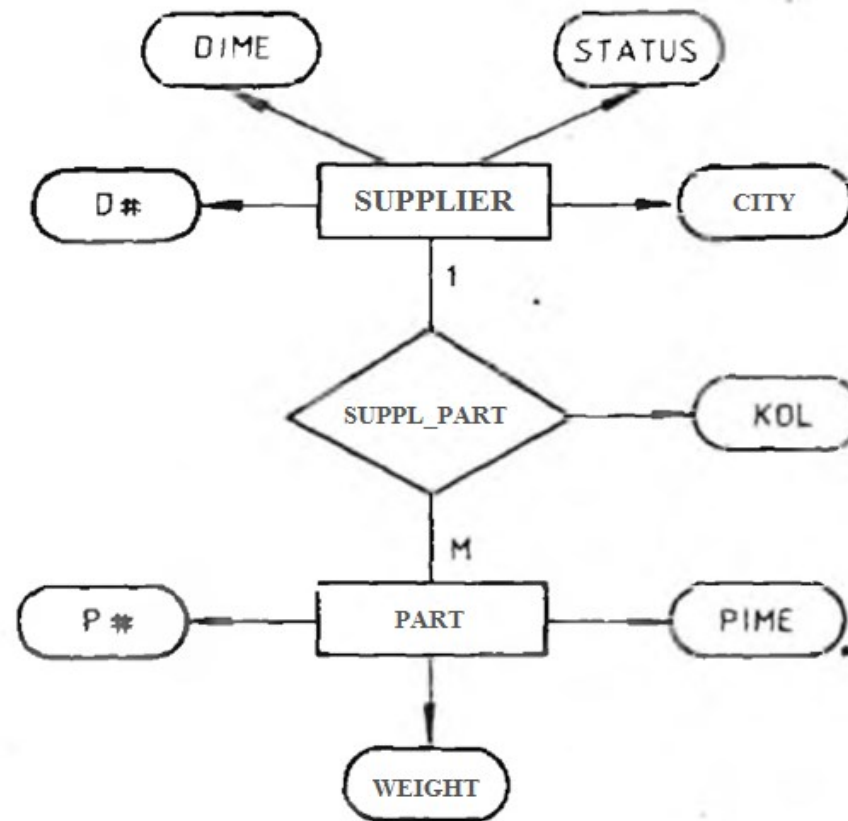
Representation of a network structure diagram as a set of two hierarchies

# “Entity – Relationship” Data models

- The Entity-Relationship (ER) data model is, in a sense, a summary of the above presented network and hierarchical data models.
- Main characteristics of the model are its expressive power and the generality with which complex subject areas can be described.
- The main structures in the ER-model are sets of entities (the entity type) and sets of relations (the relation type).
- The "entity" sets represent the structure of the object classes, and the “relation” sets represent the structure of the relations between these object classes.

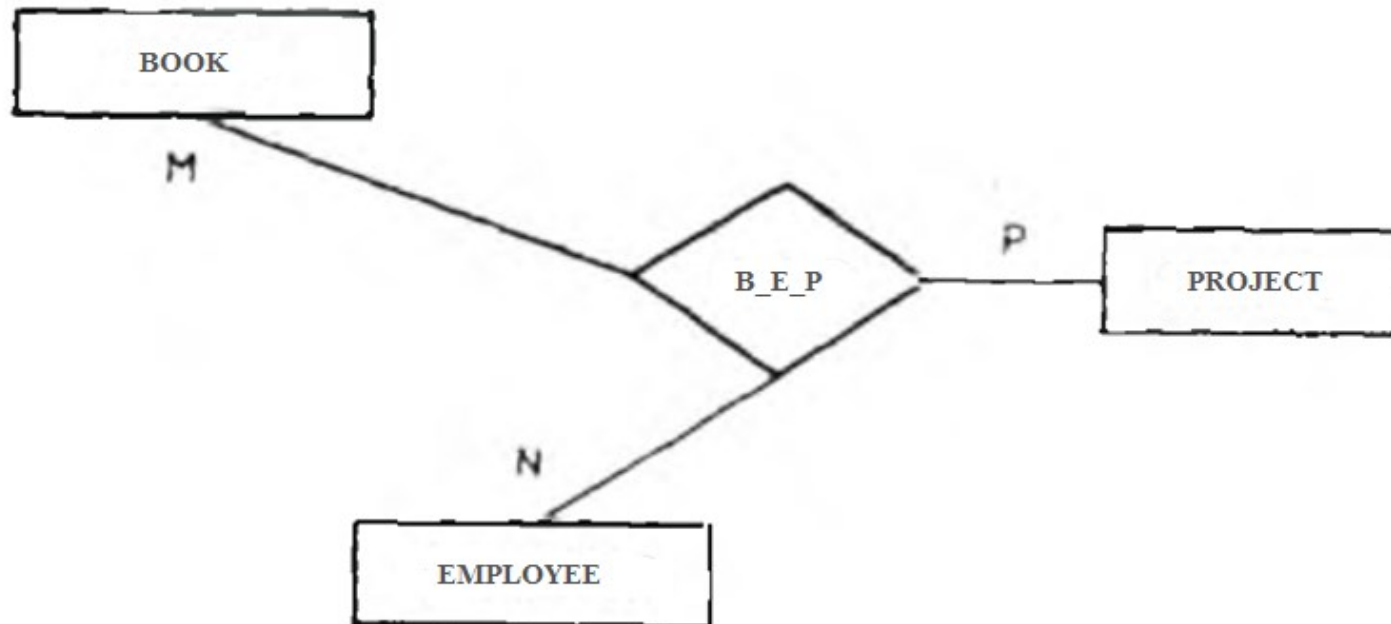


# “Entity – Relationship” Data models



ER-DIAGRAM FOR "SUPPLIES" DB

# “Entity – Relationship” Data models



Relationships of M:N type between some classes

# “Entity – Relationship” Data models

- Through the ER-model, the structure of a DB is represented by diagrams called ER-diagrams.
- These diagrams are constructed using three types of geometric shapes rectangle, rhombus and oval.
- Object classes are represented by rectangles, and their attributes are represented by separate ovals connected to the corresponding rectangles.
- The relationships between object classes are represented by rhombus, which can also have attributes.

# Questions and exercises:

1. Explain different languages present in DBMS ?
2. What is difference between DDL and DML in DBMS
3. Explain different levels of data abstraction in a DBMS.



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 1. Theoretical basis of Databases

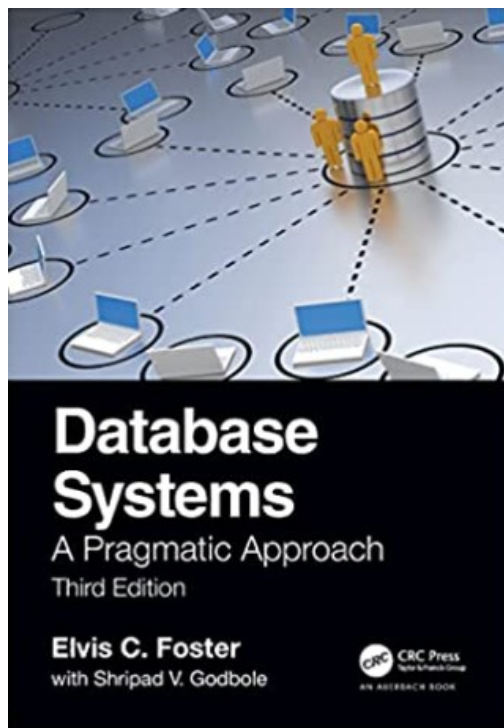
### ☐ Topic 4. Basic search methods. File types.

#### ☐ Lesson 1. Basic search methods.

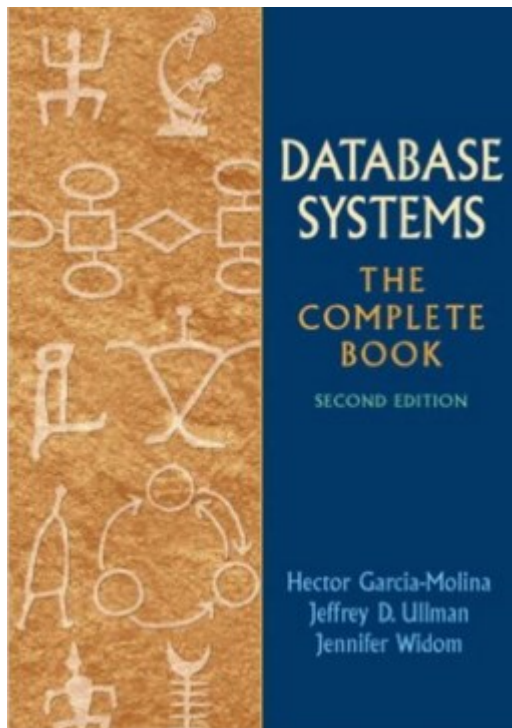


ERASMUS+

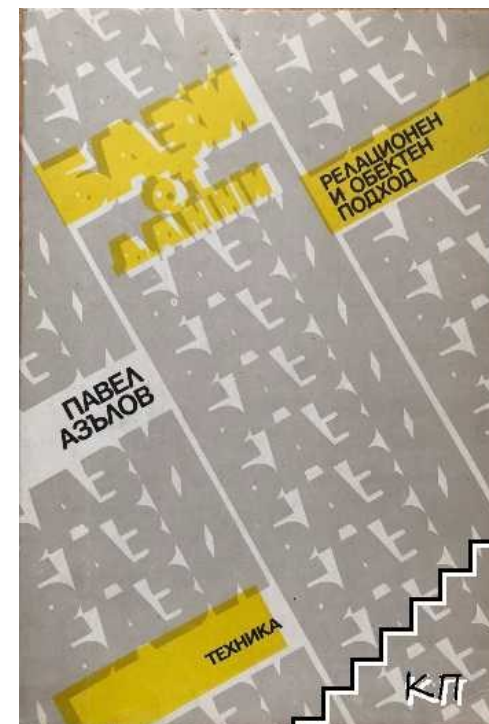
# BASIC SEARCH METHODS



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2rd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.



# Problem: Search

- We are given a list of records.
- Each record has an associated key.
- Give efficient algorithm for searching for a record containing a particular key.
- Efficiency is quantified in terms of average time analysis (number of comparisons) to retrieve an item.

# Search

[ 0 ]

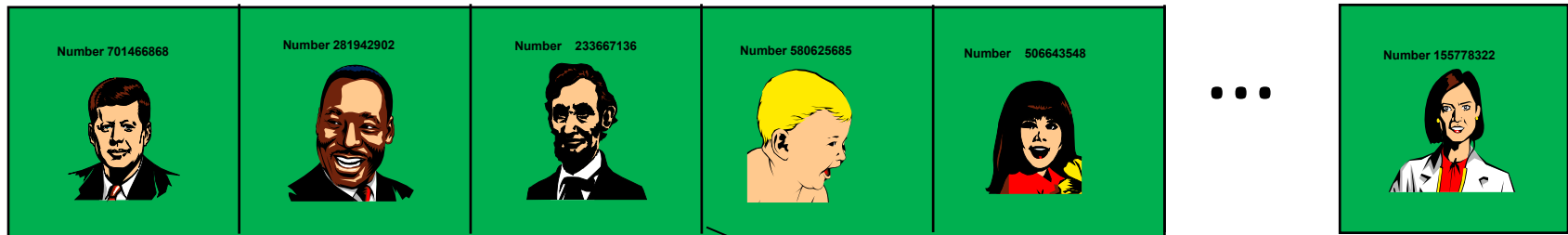
[ 1 ]

[ 2 ]

[ 3 ]

[ 4 ]

[ 700 ]



Number 580625685

Each record in list has an associated key.  
In this example, the keys are ID numbers.

Given a particular key, how can we efficiently retrieve  
the record from the list?

# BASIC SEARCH METHODS

- As stated in the previous lectures, the main task in data organization is to determine by a given key the location of the record with that key.
- In this section, some classical solutions to this task are considered, with particular attention paid to the search speed of the individual methods.

## ➤ *SEQUENTIAL SEARCH (Linear Search)*

We assume that  $F$  is a file with  $n$  records,  $n \geq 1$ . If the records of  $F$  are not sorted by the key values, then the most natural thing to do is to search for a particular record, going through the file sequentially record by record from one end to the other.

## BASIC SEARCH METHODS

- Suggest that the probability of each record being searched is the same, i. e.  $P = 1/n$  for each  $i=1,2,\dots,n$ , the total average number of checks required to find a record from the file is:

$$E = \sum_{i=1}^n iP_i = \sum_{i=1}^n i(1/n) = (1/n) \sum_{i=1}^n i = (n+1)/2$$

- Obviously, for large values of  $n$ , the estimate of  $E$  will be too large and the search for a particular record will be slow.
- Sequential search is therefore only applied if the file is unordered and/or if it is necessary to review and process all or most of the file's records.

# BASIC SEARCH METHODS

## ➤ *BLOCK SEARCH*

- The block search can be performed if the file is ordered by the key values.
- Again we assume that the file  $F$  has  $n$  records and its records are conditionally grouped into separate groups (blocks) of  $x$  records in each block.
- The last block may have less than  $x$  number of entries.
- Under these assumptions, the number of all blocks will be  $\lceil n/x \rceil$ .
- The sign  $\lceil a \rceil$  points out the smallest integer not less than  $a$ .

# BASIC SEARCH METHODS

## ➤ *BLOCK SEARCH*

- Instead of reviewing the records sequentially one after the other, it is obviously better to review only the most recent records in each block. Then the average amount of reviewed blocks is:

$$M_1 = (\lceil n/x \rceil + 1)/2$$

- If such a block search finds a block that possibly contains the record being searched, then a sequential search can be performed on that block with no more than x checks, where on average  $M_2$  checks will be made:

$$M_2 = \sum_{i=1}^{x-1} i(1/x) = (1/x) \sum_{i=1}^{x-1} i = (x - 1)/2$$

# BASIC SEARCH METHODS

## ➤ *BLOCK SEARCH*

- Thus, the total average number of checks  $E$  in the block search is:

$$\begin{aligned} E &= M_1 + M_2 = (\lfloor n/x \rfloor + 1)/2 + (x - 1)/2 = \\ &= (\lfloor n/x \rfloor + x)/2. \end{aligned}$$

- This estimate depends on the number  $x$  of records in a block. We could minimize this number by determining that value of  $x$  for which  $E$  has a minimum value:

$$E'(x) \approx (-n/x^2 + 1)/2 = 0$$

- Hence we determine,  $x \approx \sqrt{n}$  and for the minimum value of  $E$  we get:  $E(\sqrt{n}) = (n/\sqrt{n} + \sqrt{n})/2 = \sqrt{n}$



# Linear search summary

- Step through array of records, one at a time.
- Look for record with matching key.
- Search stops when
  - record with matching key is found
  - or when search has examined all records without success.

# Pseudocode for Linear search

```
// Search for a desired item in the n array elements
// starting at a[first].
// Returns pointer to desired record if found.
// Otherwise, return NULL
...
for(i = first; i < n; ++i )
    if(a[first+i] is desired item)
        return &a[first+i];

// if we drop through loop, then desired item was not found
return NULL;
```

# Linear search analysis

- What are the worst and average case running times for serial search?
- We must determine the O-notation for the number of operations required in search.
- Number of operations depends on  $n$ , the number of entries in the list.

# Worst case time for linear search

- For an array of  $n$  elements, the worst case time for serial search requires  $n$  array accesses:  $O(n)$ .
- Consider cases where we must loop over all  $n$  records:
  - desired record appears in the last position of the array
  - desired record does not appear in the array at all

# Average case for linear search

## Assumptions:

1. All keys are equally likely in a search
2. We always search for a key that is in the array

## Example:

- We have an array of 10 records.
- If search for the first record, then it requires 1 array access; if the second, then 2 array accesses.  
*etc.*

The average of all these searches is:

$$(1+2+3+4+5+6+7+8+9+10)/10 = 5.5$$

# Average case time for linear search

Generalize for array size  $n$ .

Expression for average-case running time:

$$(1+2+\dots+n)/n = n(n+1)/2n = (n+1)/2$$

Therefore, average case time complexity for serial search is  $O(n)$ .

# BASIC SEARCH METHODS

## ➤ *BINARY SEARCH*

- Binary search, as well as block search, only applies if the file is ordered.
- Let  $k$  be a positive integer for which the inequalities are valid:

$$2^{k-1} \leq n < 2^k$$

- We divide the file into two "equal" parts. The searched record is in either the first or second half, or none of them.
- This process continues until the record is found or found to be missing from the file. The estimate of the number of checks is:



# BASIC SEARCH METHODS

## ➤ *BINARY SEARCH*

- At 0 file splits there are  $n$  records.
- At 1 file split,  $n/2$  records remain.
- At 2 file splits,  $n/(2^2)$  records remain.
- ...
- At  $k-1$  file splits,  $n/(2^{k-1})$  records remain.
- But  $1 \leq n/(2^{k-1}) < 2$ . Оттук следва, че:  $0 \leq \log_2 n - \log_2 2^{k-1}$
- Finally we get:  $k_{\max} = \lfloor \log_2 n \rfloor + 1$
- Without dwelling on the derivation of the average estimate of the binary search  $E$ , we will only point out that:  $E = \log_2 n - 1$

# BASIC SEARCH METHODS

## ➤ *BINARY SEARCH*

- Here are some examples of the behavior of average scores in sequential, block and binary search.

$n$	$\sqrt{n}$	$\log_2 n - 1$
16	4	3
256	16	7
1024	32	9
$2^{2n}$	$2^n$	$2n - 1$

# BASIC SEARCH METHODS

## ➤ *BINARY SEARCH*

- The cost of faster search in block and binary search is paid for by the time it takes to maintain the file ordering. These methods are therefore usually particularly suitable for weakly variable files.
- The discussed block and binary search methods are methods typical for searching files that fit entirely in RAM. Such files are very often the indexes in index files.

# Binary Search Pseudocode

```
...
if(size == 0)
    found = false;
else {
    middle = index of approximate midpoint of array segment;
    if(target == a[middle])
        target has been found!
    else if(target < a[middle])
        search for target in area before midpoint;
    else
        search for target in area after midpoint;
}
...
```

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53



Find approximate midpoint

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53



Is 7 = midpoint key? NO.



# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53




Is 7 < midpoint key? YES.

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53



Search for the target in the area before midpoint.

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53



Find approximate midpoint

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53



Target = key of midpoint? NO.

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53



Target < key of midpoint? NO.

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53

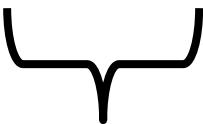


Target > key of midpoint? YES.

# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53

A black bracket is positioned below the array, starting from the middle of the array (index 2) and pointing downwards, indicating the search area for the target.

Search for the target in the area after midpoint.



# Binary Search

Example: sorted array of integer keys. Target=7.

[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]	[ 5 ]	[ 6 ]
3	6	7	11	32	33	53



Find approximate midpoint.

Is target = midpoint key? YES.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# Binary Search Implementation

```
void search(const int a[ ], size_t first, size_t size, int target, bool& found, size_t& location)
{
    size_t middle;
    if(size == 0) found = false;
    else {
        middle = first + size/2;
        if(target == a[middle]){
            location = middle;
            found = true;
        }
        else if (target < a[middle])
            // target is less than middle, so search subarray before middle
            search(a, first, size/2, target, found, location);
        else
            // target is greater than middle, so search subarray after middle
            search(a, middle+1, (size-1)/2, target, found, location);
    }
}
```

# BASIC SEARCH METHODS

BASIS FOR COMPARISON	LINEAR SEARCH	BINARY SEARCH
Time Complexity	The formula can be written as $O(N)$	$O(\log 2 N)$ is the formula that can be followed for this search
Sequential	Linear search is led by sequence; it starts from the first point and ends at the last point.	The binary search begins from the middle point.
The most compelling case time	The first Element serves to the most appropriate case time	Centre Element is the most relevant case time in this search.
Prerequisites needed for an array	Not needed	The array must be formed in sorted order
The worst-case scenario for n elements	N number of comparisons will be needed	The conclusion can be derived only after $\log_2 N$ comparisons
Capable of being implemented on	Linked lists and arrays	Incapable of being implemented directly on linked lists
Insert operation	Can be inserted with ease at the end of lists	Processing is needed to make insertions at their proper place and for the sake of maintaining sorted lists.
Algorithm type	Iterative characteristics depicted	Characteristics of “divide and conquer” features are described.
Utility	Easy to decipher and apply. There is no need for ordered elements.	The algorithms are tricky to understand and apply. The elements have to be organized in the proper manner and order.
Lines of Coding	Less	More

ERASMUS+

# Questions and exercises:

1. Which is basic search methods ?
2. What is linear search?
3. What is binary search?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 1. Theoretical basis of Databases

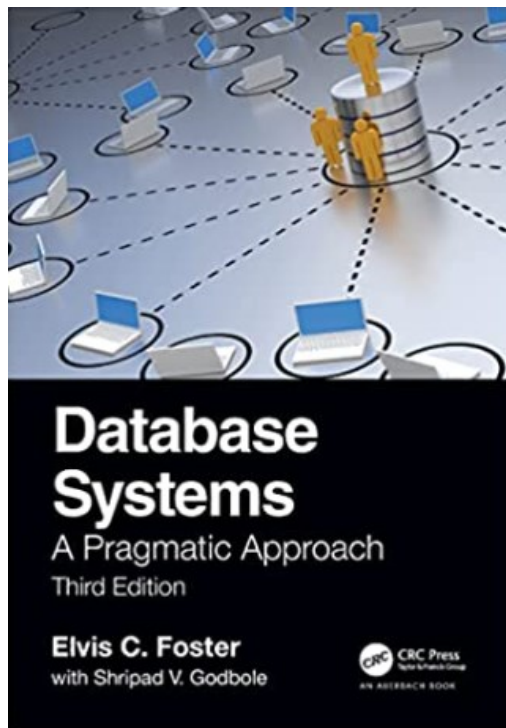
### ❑ Topic 4. Basic search methods. File types.

#### ❑ Lesson 2. Types of database files.

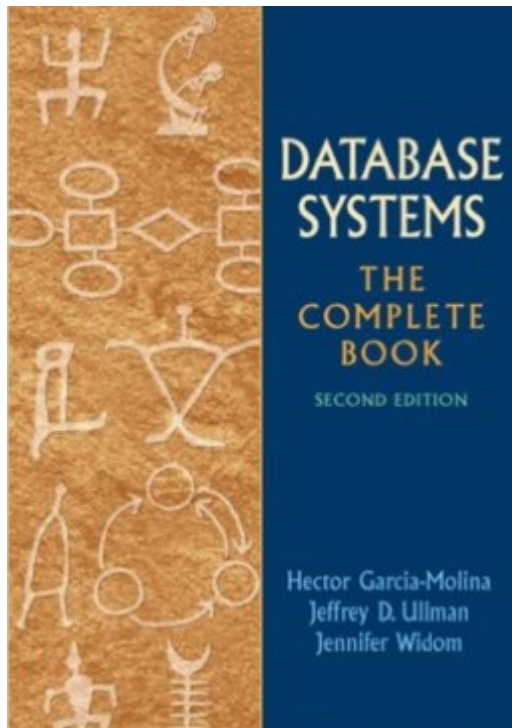


ERASMUS+

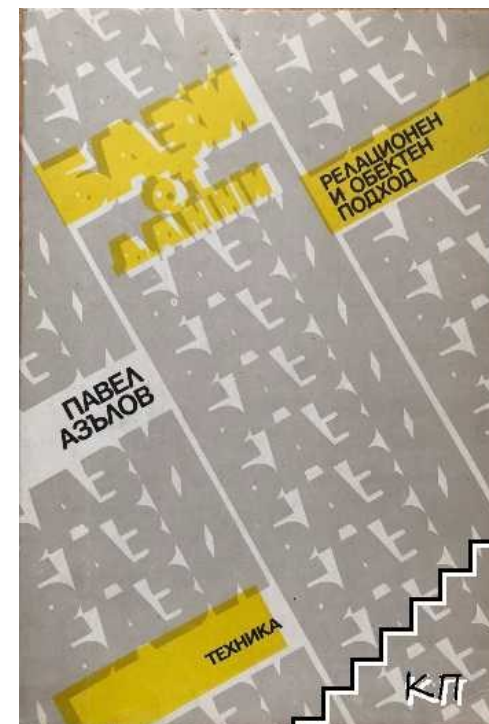
# TYPES OF DATABASE FILES



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.



# INDEX FILES

## ➤ *BASIC CONCEPTS*

We are looking at a file of records:

$$F_1, F_2, \dots, F_m, F_{m+1}, F_{m+2}, \dots, F_{2m}, \dots$$

which are arranged in ascending order of the key values, i. e.

We're creating a new file  $I(F)$ , which record type has two fields:

$$k(F_1) < k(F_2) < \dots < k(F_m) < \dots$$

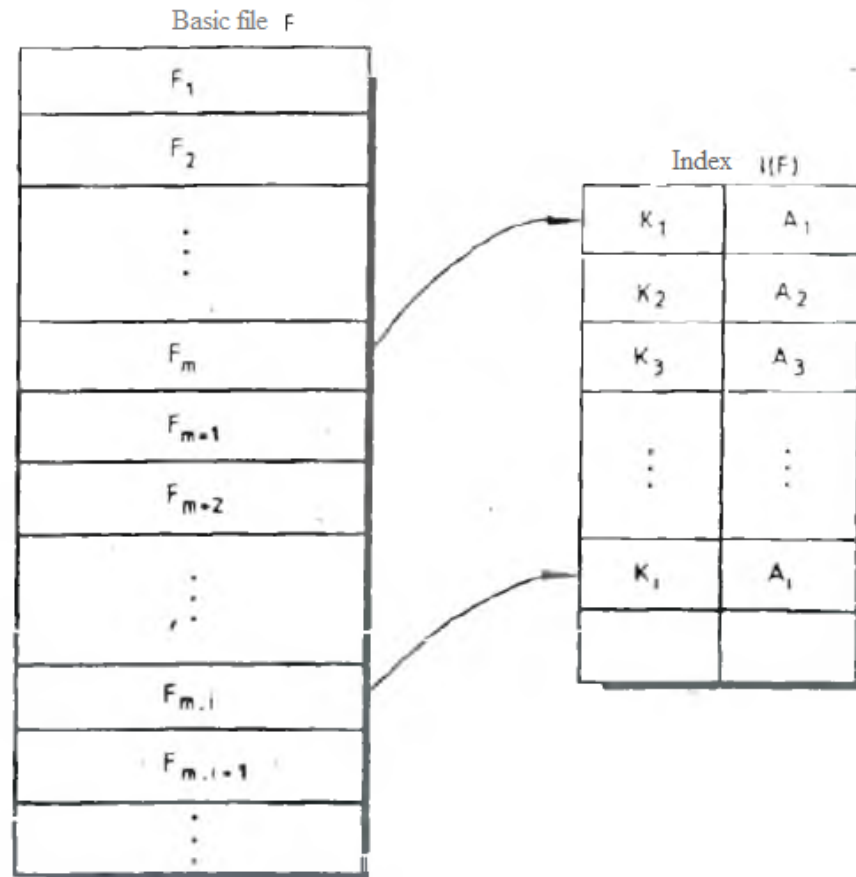
$$(K_i, A_i), \quad i=1, 2, 3, \dots$$

, where  $A$  is the address of the physical block, where records of  $i$  group are stored,  $i=1,2,\dots$ , of the basic file  $F$ .

# INDEX FILES

## ➤ *BASIC CONCEPTS*

Basic type of  
index file



# INDEX FILES

## ➤ *BASIC CONCEPTS*

- The newly retrieved file  $I(F)$  is called the index of file  $F$ , and the file  $F$  having such an index is the index file.
- In fact, the **index** is a table that can be used to create a procedure (method) to access the records in the main file.
- This access method is known as the index access method.
- Since the main file is ordered, it follows that the index will be ordered as well.
- This means that there are opportunities for an index organization where searching can be accelerated significantly.
- This fact is essential because the speed of searching in the main file strongly depends on the speed of searching in its index.

# INDEX FILES

## ➤ *INDEX-SEQUENTIAL FILE ORGANIZATION*

- If for every  $m$  records ( $m > 1$ ) of the main file, one record is created in the index, this means that the index is a file that in this case is much smaller in size than the main file.
- As a consequence, it is possible that the index could be fully or partially present in RAM, and this would indeed greatly reduce the total search time for records.
- There are cases, however, where the main file has a very large volume, hence its corresponding index also becomes very large.

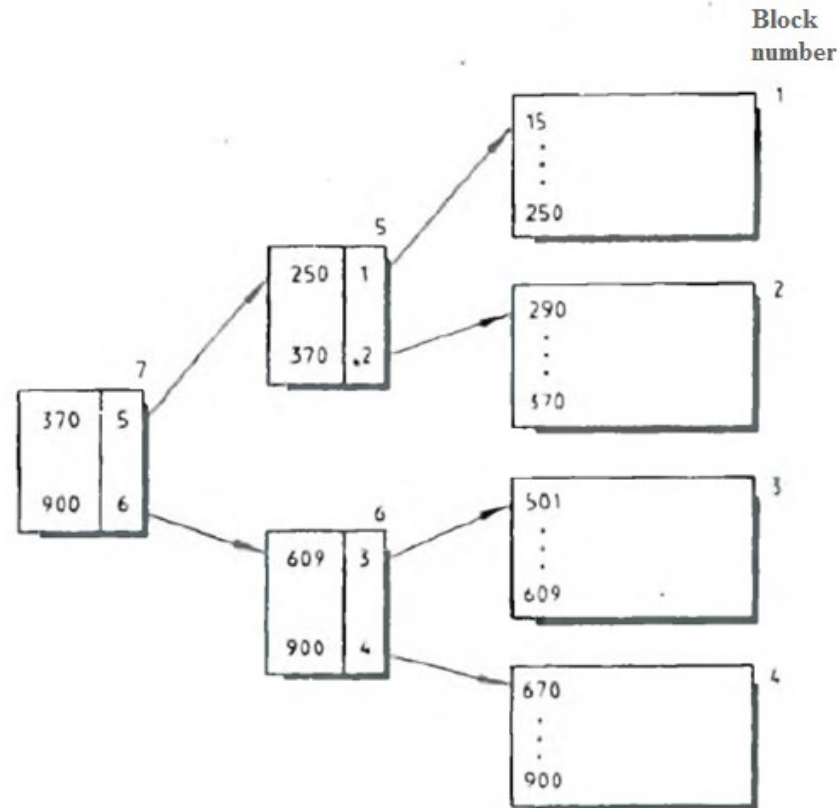
# INDEX FILES

## ➤ *INDEX-SEQUENTIAL FILE ORGANIZATION*

- Since the index is also a file, it follows that it could also be indexed and a new index could be created.
- This process of indexing the index can continue, resulting in a multi-level index.
- Searching a record in a file with such organization will be discussed with a concrete example.

# INDEX FILES

## ➤ *INDEX-SEQUENTIAL FILE ORGANIZATION*



Index-sequential organization with two levels index

# INDEX FILES

## ➤ *INDEX-SEQUENTIAL FILE ORGANIZATION*

- Let's search for a record with key  $K=576$ . The top-level index is checked first.
- In our case, this is the second-level index. For the key  $K$  of the searched record the condition  $370 < K < 900$  is satisfied.
- Therefore, the search should continue in block № 6 of the first level index.
- Since  $K < 609$ , the desired record will eventually be in block 3, where records whose keys are in the interval  $[501, 609]$  are stored.



# INDEX FILES

## ➤ *INDEX-SEQUENTIAL FILE ORGANIZATION*

- Records in index-sequential files can be read consequently from the first to the last or in a random order by setting the key of the searched index.
- Problems with index-sequential organization occurs in two cases:
  - 1. It is necessary to enter a record in a given block which is simply defined because of the organization of the main file and this block is already entirely full.**
  - 2. It is necessary even the last record of a given block to be eliminated.**

# INDEX FILES

## ➤ *INDEX-SEQUENTIAL FILE ORGANIZATION*

- The first problem could be **solved** in two ways:
  - a) an additional file is used, called overflow area;*
  - b) one part (for example a half) of records of the overflowed block are moved in a new block and this leads to creating a new file in the index.*
- In repeatedly overflowing of the blocks from the main file, the efficiency of the index-sequential access method could be severely degraded, so its reorganization becomes imperative.

# INDEX FILES

## ➤ *INDEX-SEQUENTIAL FILE ORGANIZATION*

- **Solving** the second problem is simpler and boils down to index changes only. A variety of the index-sequential organization of files is the so-called index-arbitrary organization.
- With it, the number of entries in a group is one (i.e.  $m=1$ ), which means that:

*(a) each element of the main file corresponds to an index record; such an index is called a dense index (the index in the index-consistent organization is non-dense);*

*b) the master file does not need to be ordered; this would ease the operations of adding and removing records.*

## INDEX FILES

- In database the storages structures consist of files, which are similar to the files used by operating systems.
- A data file may be used to store a relation, for example.
- The data file may have one or more index files.
- Each index file associates values of the search key with pointers to data-file records that have that value for the attribute(s) of the search key.

# INDEX FILES

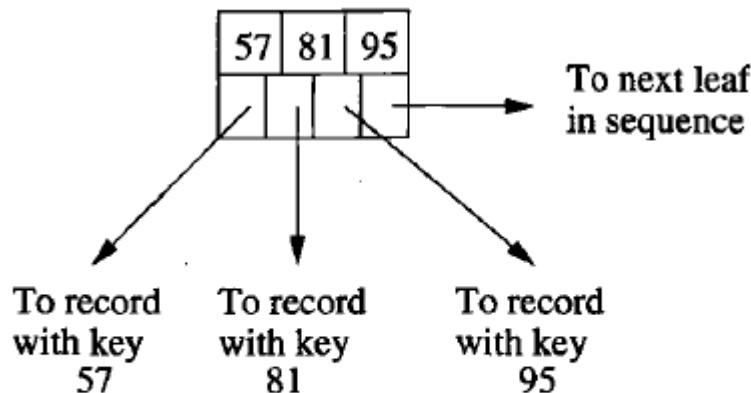
- Indexes can be “dense,” meaning there is an entry in the index file for every record of the data file.
- They can be “sparse,” meaning that only some of the data records are represented in the index, often one index entry per block of the data file. Indexes can also be “primary” or “secondary.”
- A primary index determines the location of the records of the data file, while a secondary index does not. For example, it is common to create a primary index on the primary key of a relation and to create secondary indexes on some of the other attributes.

# B-Trees

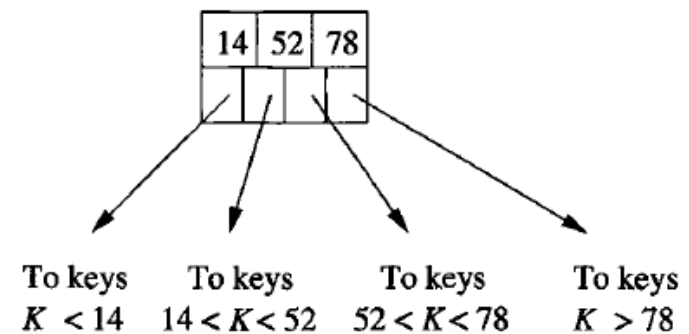
- While one or two levels of index are often very helpful in speeding up queries, there is a more general structure that is commonly used in commercial systems. This family of data structures is called B-trees, and the particular variant that is most often used is known as a B+ tree. In essence:
  - B-trees automatically maintain as many levels of index as is appropriate for the size of the file being indexed.
  - B-trees manage the space on the blocks they use so that every block is between half used and completely full.

# B-Trees

- A B-tree organizes its blocks into a tree that is balanced, meaning that all paths from the root to a leaf have the same length. Typically, there are three layers in a B-tree: the root, an intermediate layer, and leaves, but any number of layers.



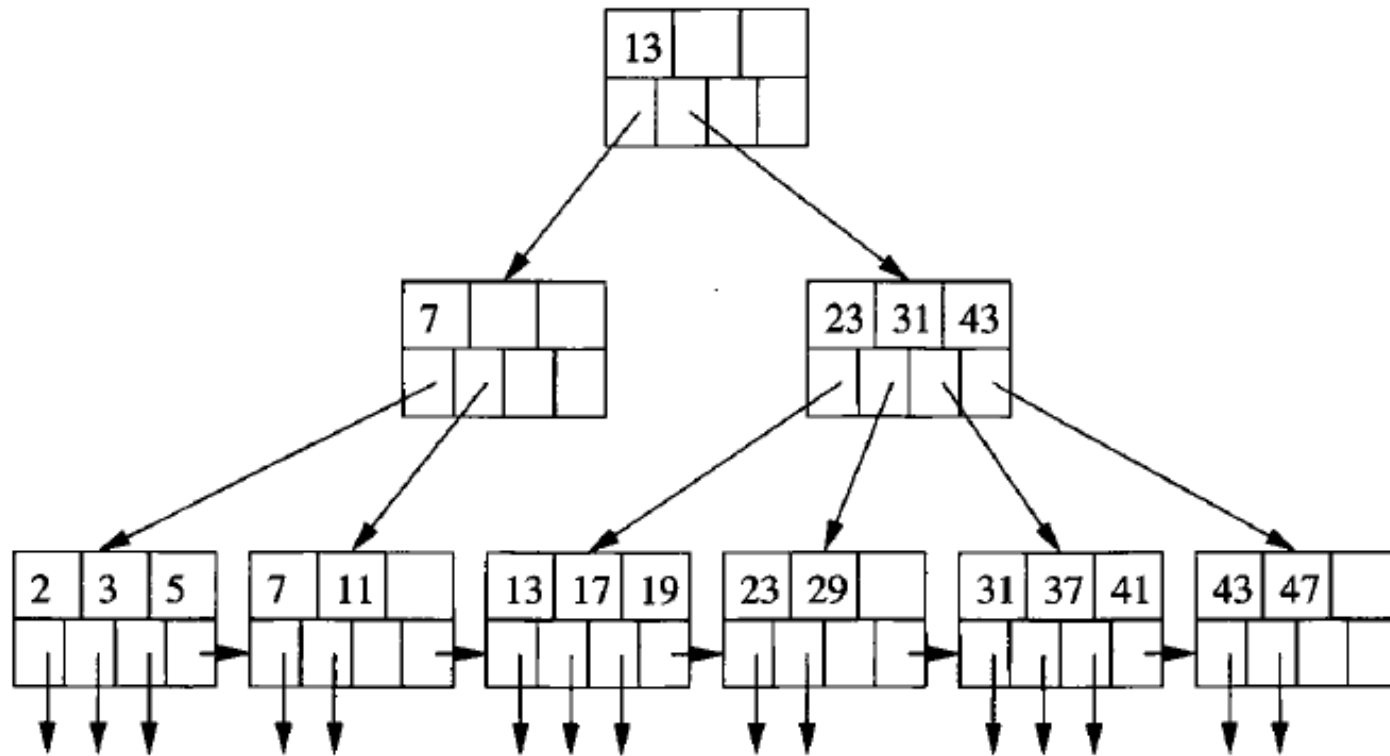
A typical leaf of a B-tree



A typical interior node of a B-tree



# B-Trees



A B-tree

# HASH-FILES

## ➤ *Basic concepts*

- Hashing is a direct access method where the key of each record is transformed into an address using a specific function.
- The set of elements, each of which is a possible key for a given file  $F$ , is called the key space and is denoted by  $S_K(F)$ .
- The set of addresses that are the starting addresses for the records of a file  $F$  is called the address space of the file and is denoted by  $S_A(F)$ .
- Any function  $h_F$ , which establishes a correspondence between the elements of  $S_K(F)$  and  $S_A(F)$ , i. e.  $h_F: S_K(F) \longrightarrow S_A(F)$ , is conventionally called a **hash function**.

# HASH-FILES

## ➤ *Basic concepts*

- Let character strings of up to 30 characters in length be used as keys for the records of an F file, and the allowed characters are uppercase letters of the Latin alphabet.
- Then the number of all elements of the space  $S_K(F)$  will be  $26^{30}$
- Each hash function "divides" the space  $S_K(F)$  into parts  $K_1, K_2, \dots, K_n$  with the following properties:

$$a) S_K(F) = K_1 \cup K_2 \dots K_n$$

$$b) K_i \cap K_j = \emptyset$$

$$b) \text{ If } h_F(k_\alpha) = h_F(k_\beta), \text{ Then such } K_i \text{ exists so that } k_\alpha \in K_i \text{ и } k_\beta \in K_i.$$

$$r) \text{ If } k_\alpha \in K_p \text{ и } k_\beta \in K_q, p \neq q, \text{ Then } h_F(k_\alpha) \neq h_F(k_\beta).$$

# HASH-FILES

## ➤ *Basic concepts*

- In other words, each hash function divides the key space into parts, each of which contains records - synonyms. The mapping of two or more record keys to the same  $S_A(F)$  address is conventionally referred to as collision.
- Files whose organization is based on hashing are often called hash files.
- The most serious problem to solve with hash files is collision resolution.
- Defining a hash function is a difficult and responsible work. It is difficult to define a universal hash function that for an arbitrary  $S_K$  space generates a minimum number of collisions.

# HASH-FILES

## ➤ *Basic concepts*

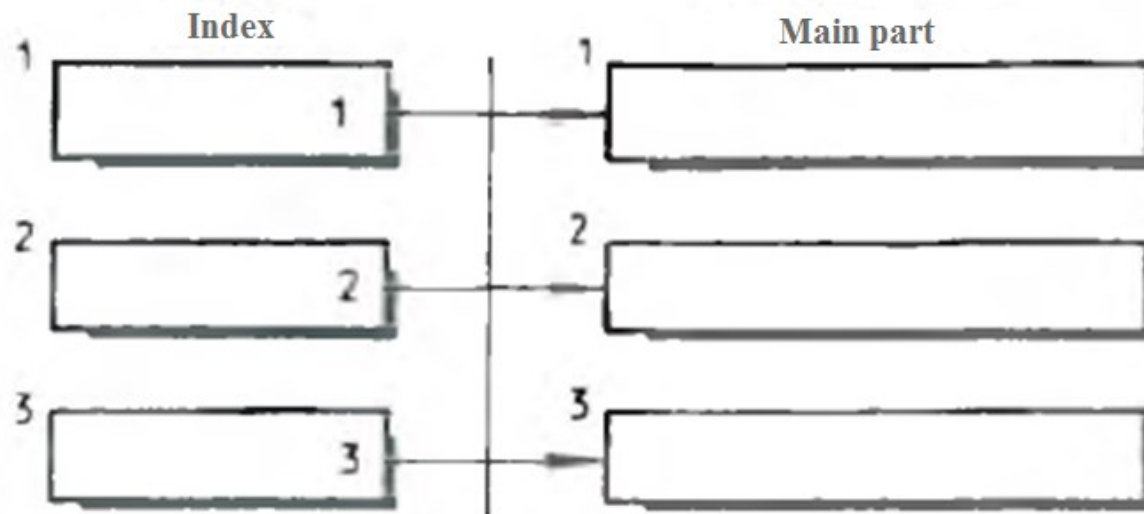
- Various hash functions are known to be suitable for certain types of keys. There are certain requirements to each hash function that are not always easy to satisfy:
  - a) to evenly and randomly distribute entries across the address space;*
  - b) all parts of the key should participate in the hash function computation, and in case of small differences in the keys, the hash function produces different addresses from the  $S_A$ ;*
  - c) the number of collisions that will occur during hashing should be as minimal as possible, and for  $n$  elements belonging to  $S_A$ , the probability that two records are synonymous should be  $1/n$ .*

# HASH-FILES

- A hash file whose size is fixed in advance and does not change during work is called a **static hash file**.
- Static hash files were the first hash files used. Their size shall be determined at their creation.
- The static hash file is made of blocks or so-called zones. Only logical records that are synonyms are stored in each zone
- When a new record is added, it is saved as the last entry in the list of synonyms from a zone. Until no zone is full, adding and searching operations are performed with a single input-output operation

# HASH-FILES

- **A dynamic hash file** (DH-file for short) consists of two parts - a main part containing the data records and an additional part called the index:



DH-file - main part and index



# HASH-FILES

- Initially, the index contains  $m$  elements, through each of which a block of the DH-file is accessed.
- $R$  records can be stored in one block (of the **DH**-file).
- With the hash function (let's denote it by  $H$ ), each of the added records is allocated to one of the  $m$  elements of the index, i. e. the hash function has the form:

$$H : K \longrightarrow \{1, 2, \dots, m\} ,$$

where  $K$  is the key space of the DH-file records.

- If  $R$  records are entered in one block, i. e. the block is completely filled, then a sequence of actions is executed, which is summarized as follows:

# HASH-FILES

1. A new empty block is added to the main part.
2. Reorganization of the records from the old block is performed, with some of them going to the newly added block.
3. The corresponding index element is modified as it grows and develops as a binary tree.

*Many different variants of dynamic hash files have been created based on the discussed hash file organization.*

# HASH-FILES

## Choice of Hash Function:

- The hash function should “hash” the key so the resulting integer is a seemingly random function of the key.
- That's improves the average time to access a record.
- Also, the hash function should be easy to compute, since we shall compute it many times.

# Questions and exercises:

1. What is file types ?
2. What is index files?
3. What is hash files ?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

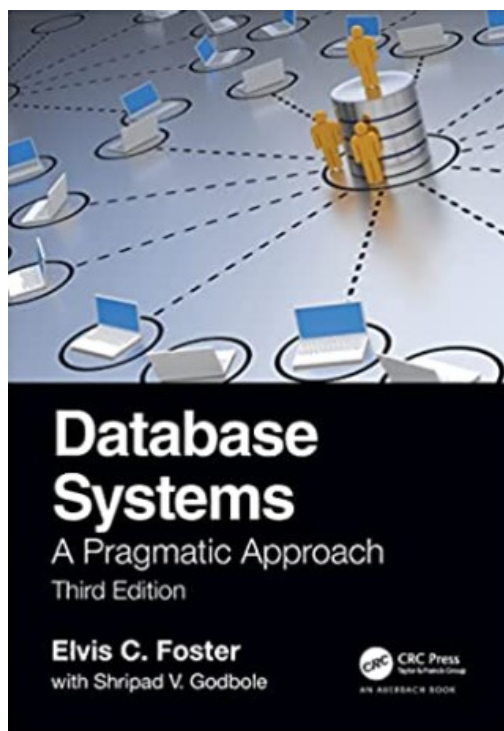
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 2. Relational approach in databases**
  - ❑ **Topic 1. Relational model - basic concepts, relational schemes.**
    - ❑ Lesson 1. Relational approach. Relational model.

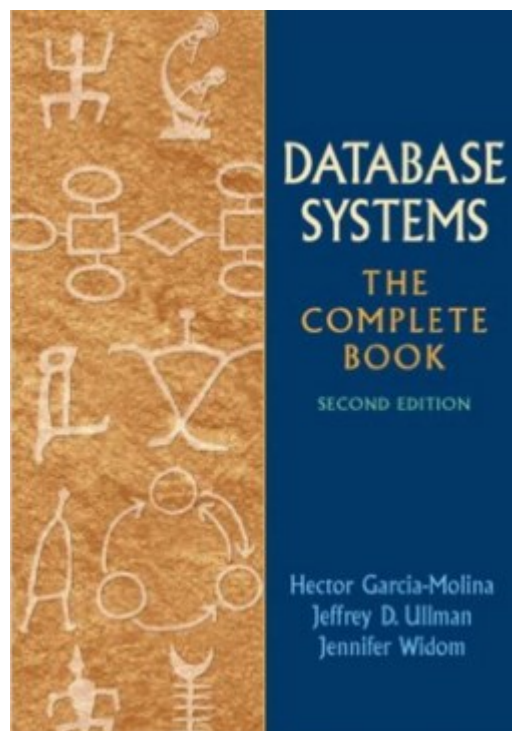


ERASMUS+

# RELATIONAL APPROACH. RELATIONAL MODEL.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.



# RELATIONAL MODEL

- The relational model started in 1970 with E. F. Codd's article "A Relational Model of Data for Large Shared Data Banks."
- The basis of the relational model is the mathematical notion of n-member relation.
- Each relation is a set of elements that consist of n components and are called n-tuples.
- In fact, a relation models a class of objects, and each tuples of the relation represents a specific object of that class.

# RELATIONAL MODEL

- The relational model gives us one way of representing the data: as a two-dimensional table called a relation.
- The following figure gives an **example** of a relation called “Movies”.

<i>title</i>	<i>year</i>	<i>length</i>	<i>genre</i>
Gone With the Wind	1939	231	drama
Star Wars	1977	124	sciFi
Wayne’s World	1992	95	comedy

Relation "Movies"

- Each of the rows represents a movie, and each of the columns represents a property of the movies.

# RELATIONAL MODEL

- The correspondence that exists between a class of objects  $R$  and its characterizing set of attributes  $A_1, A_2, \dots, A_n$  in the relational model is expressed by the record  $R(A_1, A_2, \dots, A_n)$ .
- The relational model gives the opportunity to see the DB as a set of simple (normalized) relations, which can be operated with the simplest mathematical objects - the numbers, using the operations of the so-called **relational algebra**.
- The relational model also provides a high degree of data independence as it makes no need to know the internal representation of the data and how to access it.
- An essential advantage of the relational model over other DMs is the uniform representation of object classes and the relationships between them.

# RELATIONAL STRUCTURES

- The relational model is built with three concepts: domain, attribute and relation.
- Let  $D_1, D_2, \dots, D_n$  be  $n, n \geq 1$ , sets, not necessarily distinct. Each of these sets is named, considered as a set of admissible values of some quantity, and is also called a region or domain
- Regions can be infinite or contain a finite number of elements.
- We consider the Cartesian product of the  $n$  sets  $D_1, D_2, D_3, \dots, D_n$  :

$$D_1 \times D_2 \times \dots \times D_n = \{ \langle d_1, d_2, \dots, d_n \rangle \mid d_i \in D_i, i=1, 2, \dots, n \}$$

# RELATIONAL STRUCTURES

- **For example**, if  $D_1$  denotes the signature numbers of the books in a library, i. e.

$$D_1 = \{895CIX, 1028CX, 964AIX\},$$

- and  $D_2$  denotes the numbers of employees in an institute, i. e.

$$D_2 = \{312, 205, 128\},$$

- then  $D_1 \times D_2$  will include the pairs of elements shown in the table:

# RELATIONAL STRUCTURES

Signature №	Employee №
859C1X	312
859C1X	205
859C1X	128
1028CX	312
1028CX	205
1028CX	128
964A1X	312
964A1X	205
964A1X	128

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# RELATIONAL STRUCTURES

- The meaning of the pair of elements  $\langle \text{Signature No.}, \text{Employee No.} \rangle$  could be, **for example**, "A book with signature number 859CIX has been temporarily taken by employee No. 312 “.
- As a consequence, it can be concluded that not every pair of the Cartesian product  $D_1 \times D_2$  is meaningful
- **For example**, if the book with signature number 859CIX is in a single copy and it was taken by the employee with number 312, the pairs of elements  $\langle 859C1X, 205 \rangle$  and  $\langle 859C1X, 128 \rangle$  are meaningless.
- Similar discussions can be made for the other pairs of elements.



# RELATIONAL STRUCTURES

- This brings us to the idea of considering subsets of the Cartesian product  $D_1 \times D_2$ .
- **Each subset of the Cartesian product  $D_1 \times D_2 \times \dots \times D_n$  of  $n$  regions is called a relation**
- **As an example**, consider the relation  $R \subseteq D_1 \times D_2$  of books checked out from the library at a given time

R	Signature №	Employee №
	859C1X	312
	964A1X	205
	1028CX	205

# RELATIONAL STRUCTURES

- Of course, at another point in time the content of this relation will be different.
- There are mainly two ways to represent relations.
- One of them is based on the definition of the notion of a relation, precisely that each of its elements is an ordered tuple  $\langle d_1, d_2, \dots, d_n \rangle$ , where  $d_i$  belongs to the field  $D_i$ ,  $i=1,2,\dots,n$ .
- Hence, it is clear that a relation can be considered as a table of elements in which the rows are n-tuples and the columns contain elements only from the same domain.

# RELATIONAL STRUCTURES

- Each named column of the tabular representation of a relation is conventionally called an attribute of the relation.
- Since the information content of the relation does not depend on the ordering of its attributes, it is natural to look for a second way of representation.
- In contrast to the tabular representation, the second way assumes that the columns in the table are named and therefore compliance with their ordinance is not necessary.
- Thus any relation can also be represented as a set of functions, each of which is defined in the attribute set and defines some n-tuple (element) of the relation.

# RELATIONAL STRUCTURES

- **For example**, if  $R$  is a relation with attributes  $A = \{\text{Signature No}, \text{Employee No}\}$ , it can be defined by three functions  $f_1, f_2$  and  $f_3$  of the form:

$$f : A \longrightarrow D_i, \quad i=1, 2$$

where:

$$\begin{aligned} f_1(\text{Signature No}) &= 856\text{CIX} \\ f_1(\text{Employee No}) &= 312 \\ f_2(\text{Signature No}) &= 964\text{AIX} \\ f_2(\text{Employee No}) &= 205 \\ f_3(\text{Signature No}) &= 1028\text{CIX} \\ f_3(\text{Employee No}) &= 205. \end{aligned}$$

- It is not difficult to conclude that in fact the two representations are equivalent and this allows us to use either one.

# RELATIONAL STRUCTURES

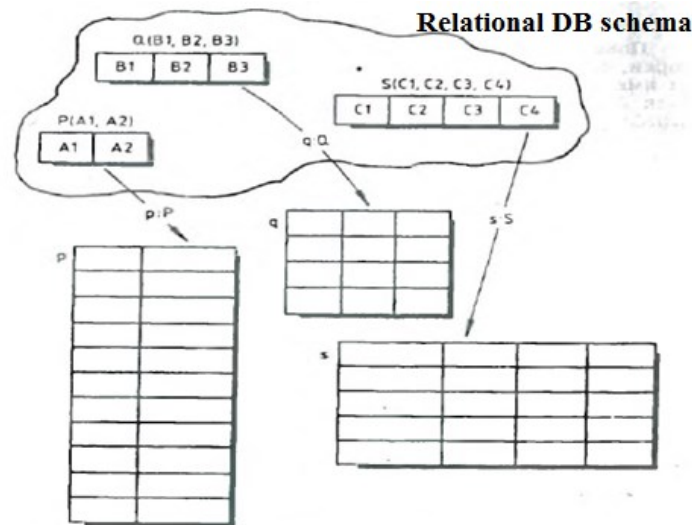
- Relations are such data structures that are variable in content and represent the current state of a class of objects at any point in time.
- The essential point in this case is that they retain their structure throughout their existence.
- The structure of each relation or, as it is also called, the type of the relation is specified by its relational schema.
- The relational schema of each relation is defined by the name of the relation and its corresponding attributes.
- **For example**, if  $A_1, A_2, A_n$ , are the attributes of an n-membered relation named R, its relational schema will be written with the expression  $R(A_1, A_2, A_n)$ .

# RELATIONAL STRUCTURES

- Since each of the attributes can only receive a value from a specific area, sometimes the more complete record  $R(A_1:D_1, A_2:D_2, \dots, A_n:D_n)$  is used when necessary.
- Let  $R(A_1, A_2, \dots, A_n)$  be a relational scheme and  $r$  be a relation with this relational scheme. This fact is briefly denoted like this:  $r:R$ .
- The set of all relational schemas that describe the object classes and the relationships between them is the DB schema.
- If  $R_1, R_2, \dots, R_k$  are the relational schemas that make up the DB schema, this is briefly written as follows:  $\{R_1, R_2, \dots, R_k\}$ .

# RELATIONAL STRUCTURES

- Each of these relational schemas has a current value and this is the content of the corresponding relation.
- Any set of concrete relations  $r_1, r_2, \dots, r_k$  for which  $r:R$  at  $i=1, 2, \dots, k$  is called the current state of the relational DB (RDB).



Relational database with three relations



# RELATIONAL STRUCTURES

- The definition of individual relational schemas is too freely, but most often the following two *rules* are followed when creating them:
  1. *Each object class is represented by a relation whose schema includes all its attributes, which become attributes of the relation as well. Each n-tuple of the relation represents a specific object (sample) of the class. The key attribute or list of key attributes of the object class, i. e. , those that uniquely identify the individual samples in the class, is taken to be the key of the relation.*
  2. *Relationships between two or more object classes are represented by a relation whose relational schema includes the key attributes of each of those object classes*

## REALTION KEY

- Since each relation is a set of non-repeating n-tuples, each relation has a key.
- It is natural for a relation to have more than one key.
- If an attribute set  $A$  is a key for a relation  $R$ , then any attribute set  $X$  of  $R$  for which  $A \subset X$  will also uniquely identify the elements of  $R$ .
- Each key is therefore also subject to a minimality requirement, which can be expressed as follows:
  1. For any two elements  $r_1 \in R$  and  $r_2 \in R$ ,  $r_1[K] \neq r_2[K]$  is valid;
  2. There is no subset of  $K$  attributes for which property 1 remains valid.

## REALTION KEY

- One of all possible keys of each relation is chosen as the key of the relation, and it is called the **primary key**.
- For the rest keys other terms are used: **probable, possible or alternative**.
- There are cases when the value of one of the components, i. e. the attribute, is unknown or **undefined** when entering a particular element in a relation. Such a value is usually called **zero**
- **The foreign key**  $K'$  for a relation  $R$  is such an attribute or list of attributes that is not a key for  $R$ , but there exists a relation  $Q$  for which  $K'$  is a primary key.
- Primary and foreign keys are basically a means of establishing connections between n-tuples of two relations.

## REALTION KEY

- In the relational model, two main integrity constraints apply:
  1. *The values of the attributes that make up the primary key cannot be zero.*
  2. *For every non-zero value of an attribute that is a foreign key, there must exist a key attribute from another relation that contains that value.*
- In addition to these two constraints, two other constraints are used
  - *the values of an attribute must be within a certain range and the value of an attribute (not necessarily a key attribute) must not be zero.*
- Another broad class of integrity constraints is introduced using functional dependencies.

# REALTION KEY

R1

Employee №	Book	
	Signature №	Date
128	723AX	15.09.89
	674CIX	18.10.89
	1021BIX	09.08.89
305	624AU	28.05.89
64	503AU	30.06.89
	372BUI	15.07.89
	1009BXI	18.07.89
	506AUI	09.08.89

R1 (Employee №, book (Signature №, date of taking))

Non-normalized relation

## REALTION KEY

R2

Employee №	Signature №	Date
128	723AX	15 09 89
128	674CIX	18 10 89
128	1021BIX	09 08 89
305	624AU	28 05 89
64	503AU	30 06 89
64	372BUI	15 07 89
64	1009BXI	18 07 89
64	506AUI	09 08 89

R2 Employee №, Signature №, Date of taking)

## Normalized relation

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



## REALTION KEY

- The relational model implies defining relations and operating on them, but not with any relations at all.
- The allowed relations for the relational model are the so-called normalized relations, i. e. , those whose relational schemes include only atomic (simple, indivisible) attributes.
- In the above two figures two relations were presented.
- One of them  $R_1$  is non-normalized, while the other  $R_2$ , expressing the same meaning and content, is a normalized relation.
- Considered also as mathematical objects, they are different. The first one is a binomial relation, and the second is a trinomial relation.
- Relational normalization is directly relevant to the design of relational DBs and these issues will be discussed in a separate lecture.



## RELATION CHARACTERISTICS

- Each relation (table) in the database has a unique name.
- Each attribute has a unique name within a given relation.
- Each relation contains unique records, it cannot contain duplicate identical records.
- There is no specific order in which records are placed in a given relation or attributes in a given record.
- Values in records are atomic - they cannot consist of different data types (data from different domains) or be the result of a calculation or concatenation.

*The physical organization of data in memory is irrelevant to the relational model, in which only the logical organization of data is valid.*

# TYPES OF REALTIONSHPIS

- To create a relation between 2 tables they must have a common (equal) field.

a) 1:1 relationship – one to one

b) 1:n relationship – one to many

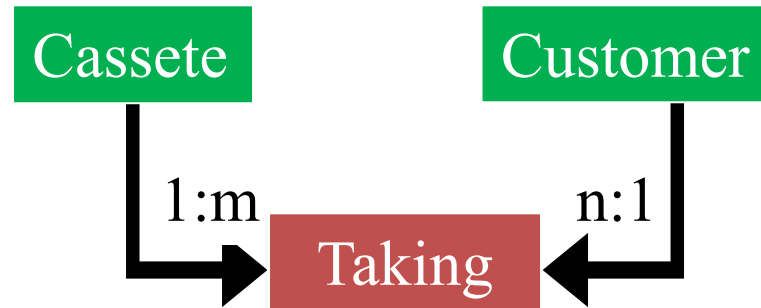
B) m:n relationship – many to many

- 1:1 relationships - one record from one table corresponds to one record from the other
  - 1 department - 1 manager
  - 1 university - 1 rector
  - 1 course - 1 course supervisor

# TYPES OF REALTIONSHPIS

- 1:n relationships type – one to many – one record from one table corresponds to many records from the other
  - 1 genre – many movies
  - 1 department – many employees
  - 1 school – many students
- m:n relationships type – many to many - one record from one table corresponds to many records from the other and vice versa.

# TYPES OF REALTIONSHPIS



Many-to-many relationships are not supported by the DBMS. They must be broken into 2 1:n relationships by adding a new table

# Questions and exercises:

1. Relational databases primarily connect multiple tables together by using which of the following?

- ✓ DDL and DML language
- ✓ Primary Key and Foreign Key
- ✓ Superkey and unique key
- ✓ Check and NULL

2. Explain different types of relationships amongst tables in a RDBMS.



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

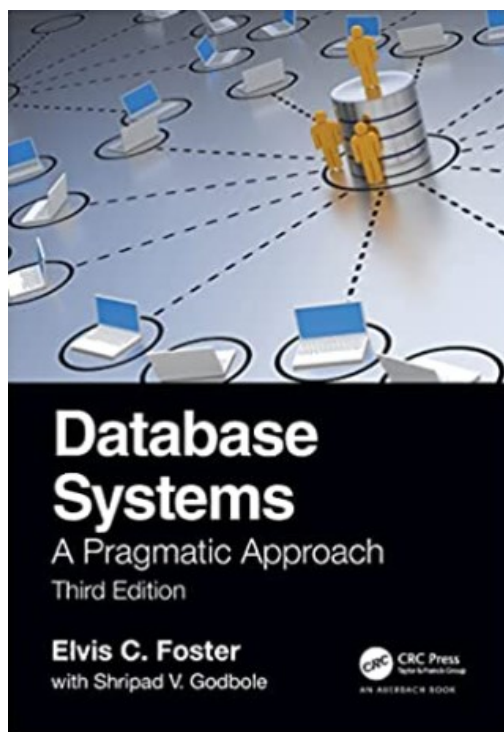
- ❑ **Module 2. Relational approach in databases**
  - ❑ **Topic 1. Relational model - basic concepts, relational schemes.**
    - ❑ **Lesson 2. Relational algebra.**



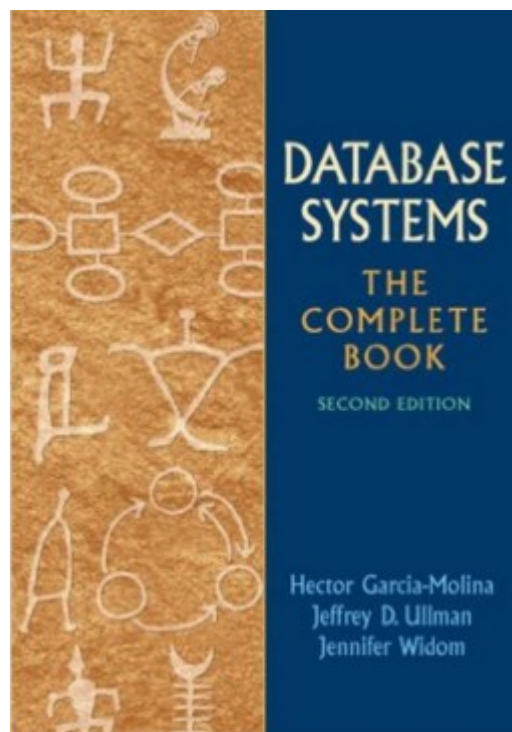
ERASMUS+



# RELATIONAL ALGEBRA



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2rd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# Introduction

- **Relational algebra** is a formal language that illustrates the basic operations for processing relational DB
- It processes corteges (records or rows) arranged in one or more relations with a single operator without organizing a cycle.
- With reservations, it can be classified as a procedural language as it indicates the way to build new relationships. It is the basis for other types of relational languages.
- **Relational calculus** is a non-procedural formal language developed by E. Codd where the user specifies what is to be contained in the result relation.

# Introduction

- This language is used to determine the capabilities of other types of relational languages.
- The operations of relational algebra proposed by CODD are divided into two groups:
  - **Multiple operations** – they are used because each relation is in general a set of corteges
    - Union;
    - Intersection
    - Difference
    - Cartesian product.

# Introduction

- The operations of relational algebra proposed by CODD are divided into two groups:
  - **Relational operations - developed specifically for DB:**
    - Select;
    - Project;
    - Join;
    - Division.

*Due to the high complexity of the above mentioned languages, practical data processing languages have been developed based on them*

# Introduction

- Let  $D_1, D_2, \dots, D_n$  be  $n, n \geq 1$  domains. We consider the list of attributes  $A_1, A_2, \dots, A_n$ , where the attribute  $A_i$  takes values from the domain  $D_i, i=1,2,\dots,n$ . We denote this list in short only by  $A$ .
- Let  $B$  be another list of  $m$  attributes  $B_1, B_2, \dots, B_m$ .
- The lists of attributes  $A$  and  $B$  are comparable if:
  1.  $m=n$
  2.  $A_k$  and  $B_k$  are of the same type for each  $k=1,2,\dots,n$ .
- Let  $p$  be an  $n$ -membered relation with relational scheme  $R(A)$ , and  $q$  an  $m$ -membered relation with relational scheme  $R(B)$ , where  $a$  and  $b$  are respectively elements of  $p$  and  $q$ , and:  $a = \langle a_1, a_2, \dots, a_n \rangle$  and  $b = \langle b_1, b_2, \dots, b_n \rangle$

## Introduction

- The concatenation of  $a$  with  $b$  is a  $(n+m)$ -torus and is defined as follows:

$$ab = \langle a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m \rangle.$$

- Under the assumptions made, by  $a.A_i$  we denote the  $i$ -th component of the element  $a$  of  $p$ . Similarly, by  $a[A_{i1}, A_{i2}, \dots, A_{ik}]$ , we denote the list of components of  $a$  whose values are respectively of  $a[D_{i1}, D_{i2}, \dots, D_{ik}]$ , i. e

$$a[A_{i1}, A_{i2}, \dots, A_{ik}] = \langle a.A_{i1}, a.A_{i2}, \dots, a.A_{ik} \rangle.$$



# Introduction

We consider 3-membered relation  $r$

$r$	Employee №	Name	Age
$x1$	235	Hristo	38
$x2$	109	Kiril	27
$x3$	203	Vasil	42
$x4$	174	Boris	40

According to the given denotations:

$$x1 = \langle 235, \text{Hristo}, 38 \rangle$$

$$x1[\text{Name}] = \text{Hristo}$$

$$x1[\text{Age, Employee №}] = \langle 38, 235 \rangle$$



# Introduction

- Except parts of rows (**elements**) of the relation, parts (**columns**) of the whole relation can be quoted. The name of the relation and the corresponding attributes are used for this purpose.

$r[\text{Name}] =$	Hristo
	Kiril
	Vasil
	Boris

$r[\text{Age, № of employee}] =$	38	235
	27	109
	42	203
	40	174

# Relational operators - UNION

- Let  $p$  and  $q$  be two  $n$ -membered relations with the same relational scheme  $R(A1, A2, \dots, An)$ .
- The union of the relations  $p$  and  $q$  is a third relation  $r$  with the same relational scheme containing the elements of  $p$  and  $q$ , i. e.

$$r = \{t \mid t \in p \cup \cup t \in q\}.$$

- A union is a two-argument operation, often denoted by the commonly used mathematical sign for union of sets "U", but sometimes also represented as a two-argument function with the following definition:

`function union( $p:R, q:R$ ): $R$`

# Relational operators - UNION

## Example

B1	B#	A	T	E	Y
	237AV	Ivanov	Geometry	Narodna prosveta Publishing house	1975
	508CX	Petrov	Informatics	Tehnika Publishing house	1985
	367CIX	Hristov	Pascal	Tehnika Publishing house	1987

B2	B#	A	T	E	Y
	508CX	Petrov	Informatics	Tehnika Publishing house	1985
	674AVI	Angelov	Basic physics	Narodna prosveta Publishing house	1973
	483AVI	Vasilev	Nuclear physics	Nauka i izkustvo Publishing house	1978
	367CIX	Hristov	Paskal	Tehnika Publishing house	1987

B=B1_B2	B#	A	T	E	Y
	237AV	Ivanov	Geometry	Narodna prosveta Publishing house	1975
	508CX	Petrov	Informatics	Tehnika Publishing house	1985
	367CIX	Hristov	pascal	Tehnika Publishing house	1987
	674AVI	Angelov	Basic physics	Narodna prosveta Publishing house	1973
	483AVI	Vasilev	Nuclear physics	Nauka i izkustvo Publishing house	1978

**B := union(B1,B2)**

# Relational operators - DIFFERENCE

- Again,  $p$  and  $q$  are assumed to be two relations with the same relational scheme  $R(A_1, A_2, \dots, A_n)$ .
- The difference of the relations  $p$  and  $q$  is a third relation  $r$  with the same relational scheme containing the elements of  $p$  not belonging to  $q$ , i. e.

`function difference( $p:R, q:R$ ): $R$ .`

- The difference of two relations is denoted by the sign " $-$ " commonly used in mathematics, and is also represented as a two-argument function with the following definition:

$$r = \{t \mid t \in p \cup t \notin q\}.$$

# Relational operators - DIFFERENCE

Example

B1	B#	A	T	E	Y
	237AV	Ivanov	Geometry	Narodna prosveta Publishing house	1975
	508CX	Petrov	Informatics	Tehnika Publishing house	1985
	367CIX	Hristov	Pascal	Tehnika Publishing house	1987

B2	B#	A	T	E	Y
	508CX	Petrov	Informatics	Tehnika Publishing house	1985
	674AVI	Angelov	Basic physics	Narodna prosveta Publishing house	1973
	483AVI	Vasilev	Nuclear physics	Nauka i izkustvo Publishing house	1978
	367CIX	Hristov	Pascal	Tehnika Publishing house	1987

B=B1_B2	B#	A	T	E	Y
	237AV	Ivanov	Geometry	Narodna prosveta PH	1975

# Relational operators - CARTESIAN PRODUCT

- Let two relations  $p$  and  $q$  be given, where  $p$  has relational schema  $P(A_1, A_2, \dots, A_m)$ , and  $q$  has relational schema  $Q(B_1, B_2, \dots, B_n)$ .
- The Cartesian product of  $p$  and  $q$  is a third  $(m+n)$  member relation  $r$ , with relational scheme  $R(A_1, A_2, \dots, A_m, B_1, B_2, \dots, B_n)$ , containing all possible concatenations of elements of  $p$  and  $q$  i. e.  
$$r = \{ab \mid a \in p \cup b \in q\}.$$
- The Cartesian product of two relations is denoted by the sign " $\times$ " commonly used in mathematics, but is also represented as a two-argument function with the following definition:

`function product( $p:P, q:Q$ ): $R$`

# Relational operators - CARTESIAN PRODUCT

Example

B	B#	A	T
	237AV 508CX 367CIX	Ivanov Petrov Hristov	Geometry Informatics Pascal

B_E	B#	S#	Date
	508CX 367CIX	92 17	15.06.89 08.10.89

BBS=BxB_E	B#	A	T	B#	S#	Date
	237AV	Ivanov	Geometry	508CX	92	15.06.89
	237AV	Ivanov	Geometry	367CIX	17	08.10.89
	508CX	Petrov	Informatics	508CX	92	15.06.89
	508CX	Petrov	Informatics	367CIX	17	08.10.89
	367CIX	Hristov	Pascal	508CX	92	15.06.89
	367CIX	Hristov	Pascal	367CIX	17	08.10.89

$BBS := \text{product}(B, B\_E)$



## Relational operators - Projection

- Let  $p$  be an  $n$ -membered relation with relational scheme  $P(A_1, A_2, \dots, A_n)$ .
- The projection of the relation  $p$  with respect to the attributes  $A_{i1}, A_{i2}, \dots, A_{ik}, i \leq k \leq n$  is a  $k$ -membered relation  $r$  with relational schema  $R(A_{i1}, A_{i2}, \dots, A_{ik})$ , which is obtained from  $p$  by removing all attributes that are not configured in the schema  $R$  and in which the repeated elements (rows) are represented only once, i. e

$$r = \{ \langle b_1, b_2, \dots, b_k \rangle ; (\exists a), a = \langle a_1, a_2, \dots, a_n \rangle, a \in p \cup a_{i1} = b_1, a_{i2} = b_2, \dots, a_{ik} = b_k \} .$$

## Relational operators - Projection

- In accordance with accepted notations, the projection  $r$  of the relation  $p$  can also be written as follows:

$$r = p[A_{i1}, A_{i2}, \dots, A_{ik}].$$

- Its expression by a function has the following definition:

function project( $p:P, A_{i1}, A_{i2}, \dots, A_{ik}:Attr$ ): $R$ .

# Relational operators - Projection

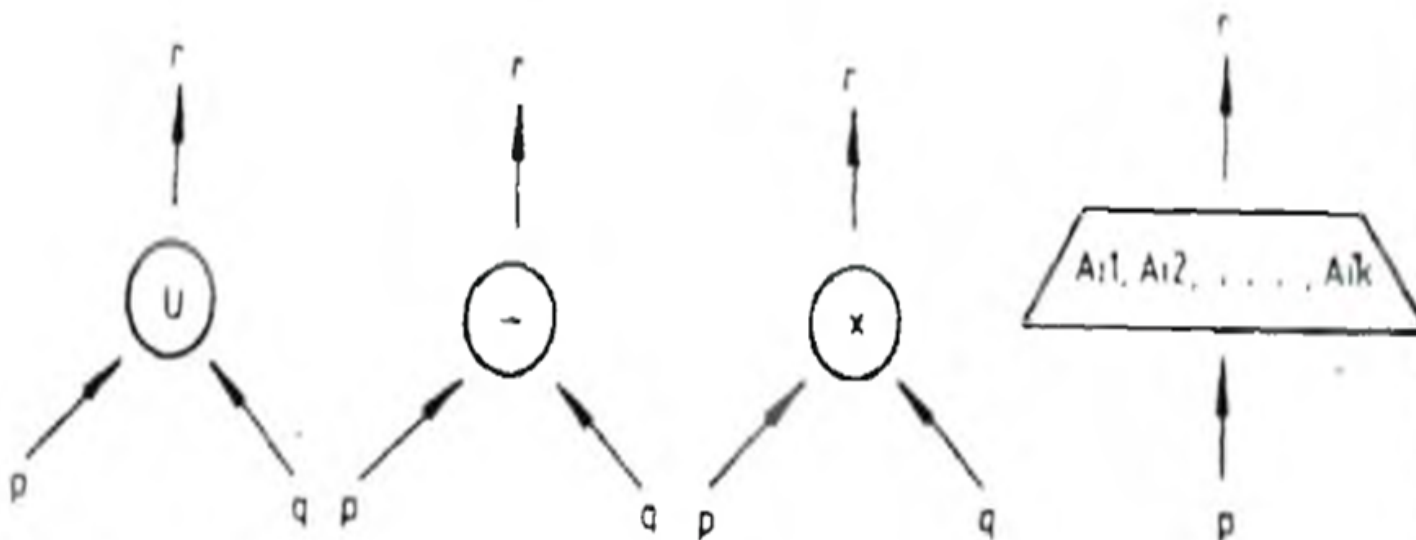
## Example

B	B#	A	T	E	Y
	237AV	Ivanov	Geometry	Narodna prosveta	1975
	508CX	Petrov	Informatics	Tehnika	1985
	367CIX	Hristov	Pascal	Tehnika	1987
	792BVI	Petrov	Trygonometry	Narodna prosveta	1985

$B1 = \pi_{A,Y}(B)$	A	Y
	Ivanov	1975
	Petrov	1985
	Hristov	1987

$B1 := \text{project}(B, A, Y)$

# GRAPHIC INDICATIONS



**graphic indication  
of the operation  
"union"**

**graphic indication  
of the operation  
"difference"**

**graphical notation  
of the operation  
"Cartesian  
product"**

**graphical notation  
of the operation  
"Projection"**

## Relational operators - RESTRICTION

Let  $p$  be an  $n$ -membered relation with relational scheme  $R(A_1, A_2, \dots, A_n)$ .  $P$  relation restriction regarding a given condition  $F$  is another relation  $r$  with the same relational scheme, each element of which fulfill the condition  $F$ , i.e.

$$r = \{ t \mid t \in p \wedge F(t) = \text{true} \}$$

It's obvious that  $r \subseteq p$ . The restriction is denoted by  $r = \sigma_r(p)$ .

And as a function has the following definition:

function restriction ( $p:R, F:\text{Expression}$ ) :  $R$ .

## Relational operators - SECTION

The section of two relations  $p$  and  $q$  with mutual relational scheme is third relation  $r$  with the same relational scheme where:

$$r = \{t \mid t \in p \cup t \in q\}$$

It is denoted with the generally accepted in mathematics sign “ $\cap$ ”. The section is also written as a function with the following definition:

$$\text{function intersect } (p, q: R): R$$

By the definitions of section and difference it follows:

$$r = p \cap q = p - (p - q) = q - (q - p)$$

## Relational operators - QUOTIENT

Let two relations  $p$  and  $q$  be given,  $q$  - non-empty relation with relational schemes  $P(A_1, A_2, \dots, A_K, B_1, B_2, \dots, B_M)$ . The quotient of  $p$  and  $q$  is third relation  $r$  with a relational scheme  $R(A_1, A_2, \dots, A_K)$  and for each  $1 \leq r$  and each  $s \in q$ ; the concatenation  $ts$  is a  $p$  element. The quotient of two relations  $p$  and  $q$  is written  $p : q$  and may be expressed by three of the main operations of relational algebra in the following way:

$$p : q = \pi_A(p) - \pi_A((\pi_A(p) \times q) - p)$$

Where  $A$  shows the list of attributes  $A_1, A_2, \dots, A_K$ .



# Relational operators - QUOTIENT

The graphical notation of the quotient of two relations is similar to the other two-argument operations by placing the ":" sign in the circle

## Example

B_E	B#	S#
	546AX	72
	328CX	47
	328CX	43
	607BV1	63
	459AV	37

S	S#
	43
	47

B_E : S	B#
	328CX

## Conclusion

- *Users search information in the relational database by making queries written in one of the formal notations for expressing operations on relations. These notations are two types:*
  1. *An algebraic notation called relational algebra, where queries are expressed by applying special operations to relations.*
  - 2 *A logical notation called relational calculus, where queries are expressed by writing logical formulas that must satisfy the tuples in the result. The two notations (subject to some restrictions) have equivalent expressive power, i. e. each can express any arbitrary query that the other can.*
- *Relational algebra is a notation for describing queries to relations.*
- *The operations of relational algebra are formally performed on relations, which are expressed as sets of tuples.*

# Questions and exercises:

1. What is Relational Algebra?
2. How can you distinguish between Relational Algebra and Relational Calculus?



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 2. Relational approach in databases

### ❑ Topic 2. Relational languages.

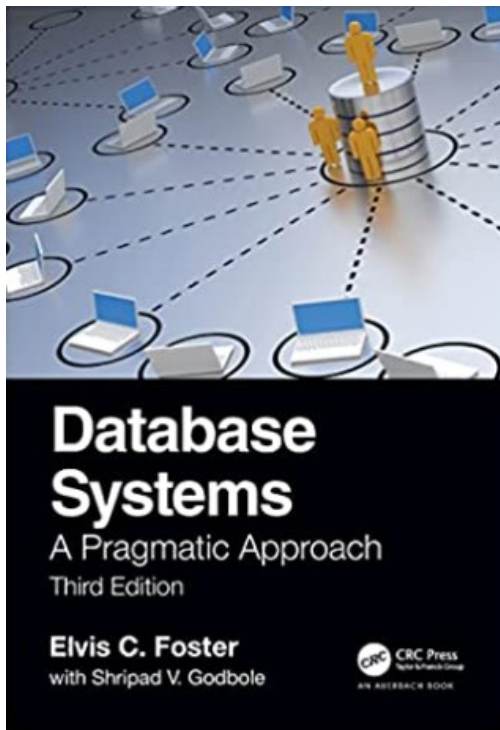
#### ❑ Lesson 1. Relational languages. Types of Relational Languages.



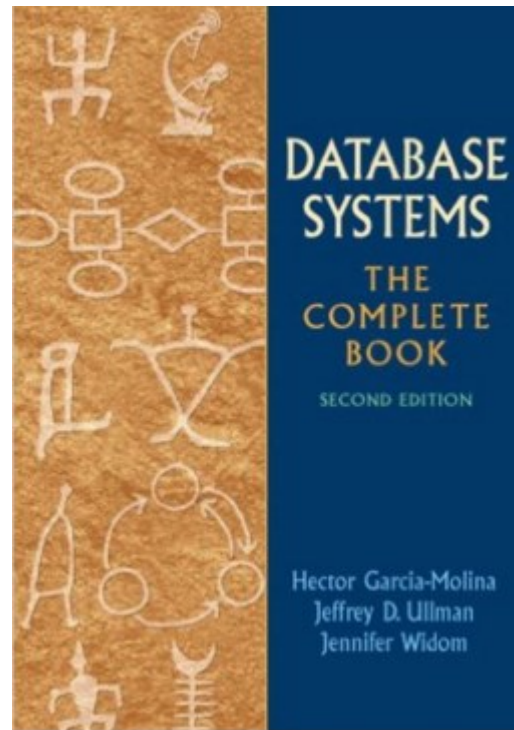
ERASMUS+

# RELATIONAL LANGUAGES.

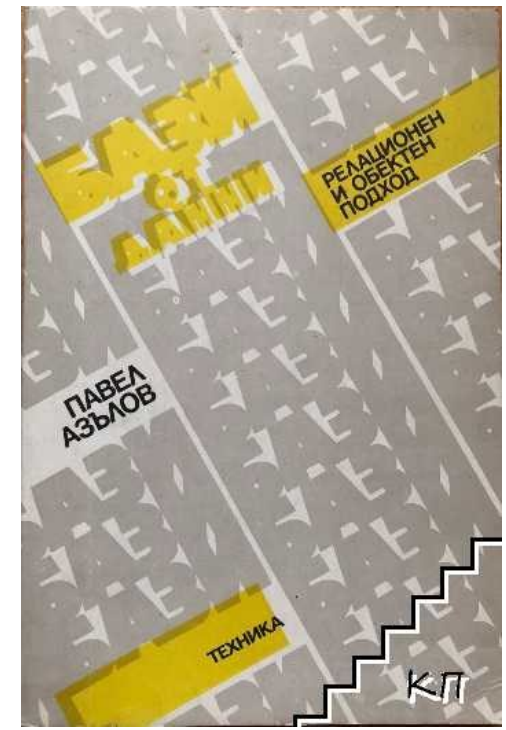
## TYPES OF RELATIONAL LANGUAGES.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.



# Introduction

- Relational languages are languages for describing and processing data in relational DBMSs.
- They are also called data languages, and because they often are incomplete and embedded within other languages, they are also called data sublanguages.
- Relational languages are divided into two main categories - languages of *relational algebra* and languages of *relational calculus*.
- Languages of relational algebra are procedural languages which, by means of the relational operations discussed in the previous chapter, allow a step-by-step description of the computational process leading to the relational answer.



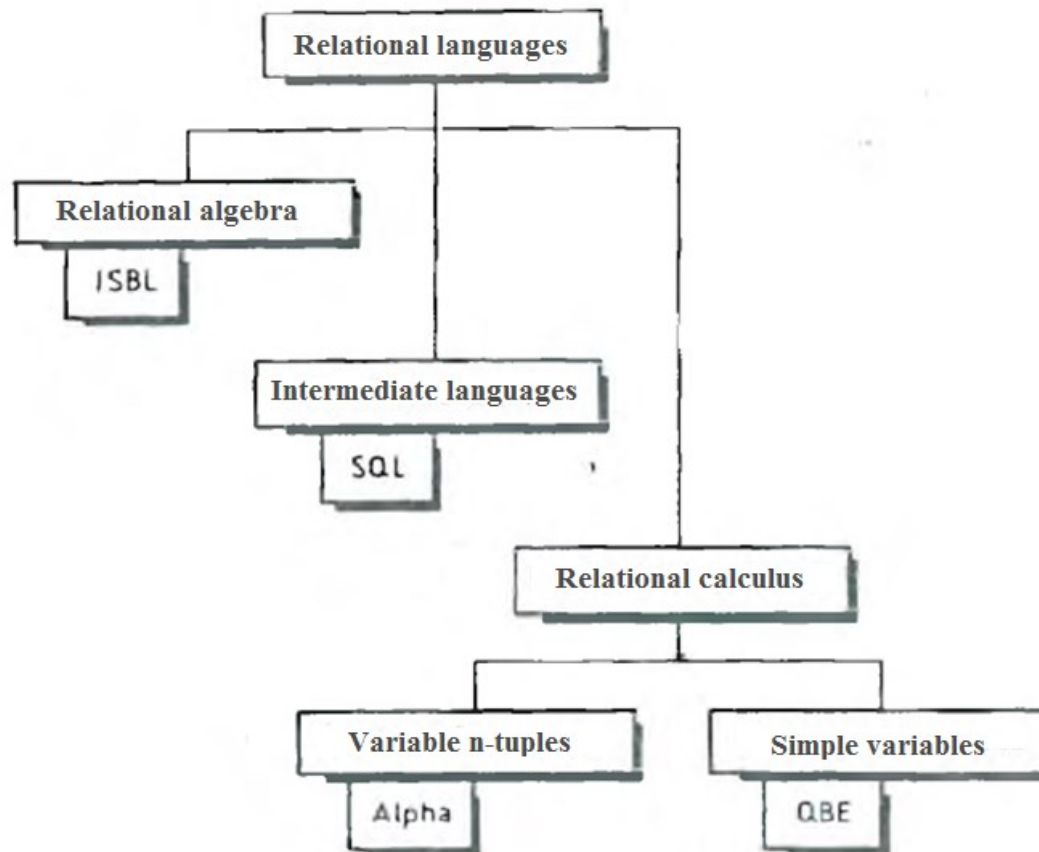
# Introduction

- *Relational calculus languages*, also called predicate languages, allow to write the general form of the relation-response without specifying the individual operations and their sequence.
- Relational calculus languages can be divided into two groups depending on the type of variables allowed in their language constructs.
- In the first type of languages, variables are used whose allowed values are elements of a relation, i. e. n-tuples.
- That's why they are called languages of relational calculus with variable n-tuples.

# Introduction

- One of the most common languages of this type is the **Alpha language**.
- The other type of languages use variables whose allowable values are elements of separate domains, i. e. simple variables, and are therefore called languages of relational calculus with **simple variables**.
- Representative of this type languages is the **QBE** language.
- Except the two main categories of languages, there are also the so-called intermediate relational languages that have the capabilities of the relational algebra and *relational calculus* languages. The language of this type that is mostly used is the **SQL** language.

# Introduction



**Relational languages categories and their representatives**

# ALPHA LANGUAGE

- The Alpha language is based on predicate calculus and its author is E. Codd.
- Language constructions are few, but powerful and expressive.
- The author of the language assumes that **Alpha** is embedded as a data sublanguage in some programming language, such as **PL/1**, **COBOL** or **FORTRAN**, whose tools are used to describe the objects that make up the DB.
- Using the language tools, the user extracts data from the DB and it is automatically placed in the workspace.
- When data is entered into the DB, it must first be located in some work area.

# ALPHA LANGUAGE

- In other words, the workspace is an intermediate link between the user and the DB.
- The user can use several workspaces at the same time, quoting them with their names.
- Each workspace automatically obtains the type (schema) of the relation that is obtained as a result of executing some relational operator.
- The operator for searching and retrieving data from the DB has the form:

`get W[(N)](L)[KS]`

# ALPHA LANGUAGE

get  $W[(N)](L)[KS]$

And the meaning of the symbols used is as follows:

get -	reserved word of the language
$W$ -	working area name
$[...]$ -	the expression in these brackets is optional and may be absent
$N$ -	the maximum number of elements in the working area $W$ ; by default, it is assumed that $W$ will contain the whole relation - reponse
$L$ -	the target list specifying the relation-response scheme; the list can contain attribute names or a relation name
$KS$ -	the criterion for selecting data and/or setting the order of the elements of the relation-response

# ALPHA LANGUAGE

- The database is **updated** when:
    - creating new relations or adding elements to existing relations;
    - removing relations from the DB or elements from given relations;
    - changing the values of some attributes in certain relations.
- These operations are illustrated by an example.

## Example

**A new element to be added to relation BOOK**

**<881 CIX, Ivanov, Database, Tehnika, 1989>**



# ALPHA LANGUAGE

**Solution:**

**First, by means of the used programming language the components of this element are entered in W workspace**

```
W.B# := '881CIX';  
W.A := 'Ivanov';  
W.T := 'Database';  
W.E := 'Tehnika';  
W.Y := 1989;
```

**The transfer of the element from the workspace W to the DB is done with the operator: put W (BOOK)**

# ALPHA LANGUAGE

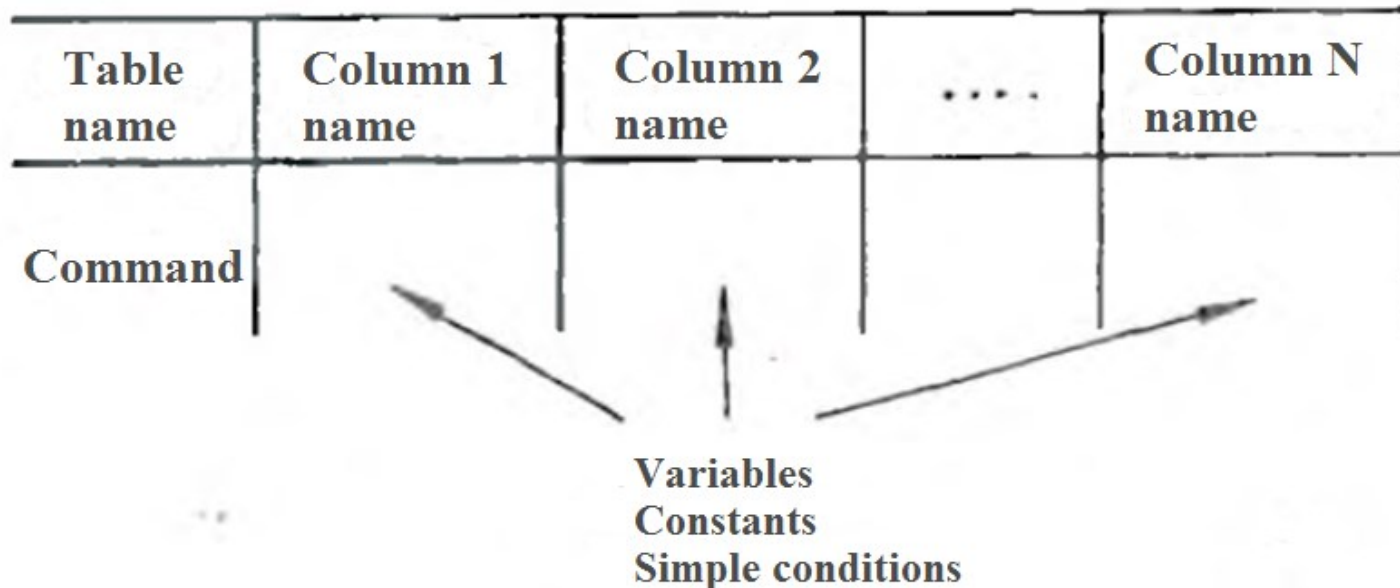
- The **Alpha** language offers several built-in features. They are used together with the get operator and specifically they can participate in both the whole list and the selection criterion. Among the more important functions are:
  - a) **count** (attribute name) - finds the number of different values in the specified attribute;
  - b) **total** (attribute name) - finds the sum of the elements in the specified attribute;
  - c) **top** (n, attribute name) – this is a Boolean function. Its value is "true" only if the value of its "attribute name" parameter is equal to the n-th largest value of that attribute. If the attribute has as values the numbers {5,3,8,4,6,7,10}, then the third largest value in order is the number 7 and the third smallest value in order is the number 5.
  - d) **bottom** (n, attribute name) – the function is the same as the function top, but refers to the n-th smallest value.

# QBE LANGUAGE

- The full name of the QBE language is Query By Example.
- The most characteristic feature of this language is its two-dimensional syntax, that's why it is used only by terminal.
- Using a special screen editor, the user creates and then uses blank tables (relations), which actually define relational schemas.
- Some authors call such an empty table a "skeleton table", a "structure of table" or a "shape of table".

# QBE LANGUAGE

- Each form contains the name of the table and the names of its columns (attributes).



**Table form with N-columns**

# QBE LANGUAGE

- **QBE** is a relational database management language and enables the definition of relational (form), update and search data in the database.
- What is important in this case is that all these types of operations are performed by a unified method, using forms.

The main principles and ideas in QBE language can be summarized as [4, 6, 16, 106]:

1. The user should be facilitated as much as possible when using the language.
2. The language syntax should be simple but with great expressive power.
3. Communication with DB is made by “filling” the forms with the appropriate “sample” row that defines the type of all desired rows.

# QBE LANGUAGE

- The search and retrieval of data from the DB is done by specifying a sample answer in the appropriate form. The example includes names of variables and constants that are placed in the appropriate columns of the form.
- For example, if you were to print out the numbers of all the projects in computer science, this could be done as follows. Enter the name of the **PROJ** table and the form appears on the screen:

PROJ	P#	S#	D	C
------	----	----	---	---

# QBE LANGUAGE

- Now the query itself should be prepared. It contains two key elements:
  - the constant 'informatics';
  - printing of the project number.
- These two elements are written in columns *D* and *P#*.

PROJ	P#	S#	D	C
	P.p1		Informatics	



# QBE LANGUAGE

- This record means the following:
- Print the project numbers, such as *p1*, which are in the field of informatics.
- The sample element *p1* is underlined and the text constant 'informatics' is not underlined.
- The printing of the project numbers is done by the P. (Print) command preceding the sample element *p1*.
- The example element "*p1*" is chosen quite arbitrarily and it is possible that no such element exists at all.
- Here is an example:

# QBE LANGUAGE

**Example**

**Extract publishers which books are used by more than one employee**

**Solution:**

BOOK	B#	A	T	E	Y
	<u>Z</u>			Tehnika P.	

B_E	B#	S#	D1	D2
	<u>Z</u>	. <u>X</u>		
	<u>Z</u>	≠ <u>X</u>		

# QBE LANGUAGE

The following operations are used to update a database: U - update, I - insert and D - delete.

Here is an example for their implementation

**Example**

Add a new row in table PROJ:

`<p8,s9, matematis ,18x>`

**Solution:**

PROJ	P#	S#	D	C
I.	p8	s9	matematis	18x

# QBE LANGUAGE

**Example**

**Combine tables BOOK and B (suppose that B has the same relation scheme as BOOK)**

***Solution:***

B	B#	A	T	E	Y
	<u>B1</u>				

BOOK	B#	A	T	E	Y
1.	<u>B1</u>				

# QBE LANGUAGE

**Example**

**Reduce the sum for project number p1 by 5 thousand**

***Solution:***

PROJ	P#	S#	D	C
	p1			<u>X</u>
U.	p1			<u>X</u> -5

# QBE LANGUAGE

The data dictionary is set of descriptions, consisting of some system tables. Three of them are described here:

TABLE – a table containing information about the DB tables;

COLUMNS – a table containing information about DB attributes;

AUTHORITY – a table, containing information about users and their rights of access to different relations. The type of access is described in details: for extracting, updating, adding or deleting data from DB.

# QBE LANGUAGE

**A new table is defined by operation I (Add)**

**Example**

**Create table PROJ.**

***Solution:***

**Begins with an empty table of the type:**




# QBE LANGUAGE

If necessary new columns are added automatically and some of them can be expanded:

I.PROJ	I.	P#	S#	D	-C

Operation P is used to enter characteristics in each column:

I.PROJ	I.	P#	S#	D	C
P. <u>X</u>					

# QBE LANGUAGE

The result of the operation is

PROJ	P#	S#	D	C
TYPE I.	string	string	string	integer
LENGTH I.	3	4	16	4
KEY I.	Y	N	N	N
DOMAIN I.	NUMBER	NUMBER	SCIENCE	MONEY
SYSNULL I.		-	-	0

The meaning of each characteristic is:

**TYPE** - defines the type of data in the column:  
char, string, real, integer, boolean;

**LENGTH** - defines the table field length (in number of positions)

**KEY** - specifies the key (Y) and non-key (N) attributes in the table

**DOMAIN** - contains the fields names which give values to the attributes

**SYSNULL** - defines the zero value to each attribute

# QBE LANGUAGE

**Example**

**Get the names of the tables containing columns  
named B#**

***Solution:***

P.	B#

**The answer is:**

BOOK	

B_E	

# QBE LANGUAGE

- Products such as Microsoft Access and Microsoft SQL Server Enterprise Manager employ the aspects of QBE.
- After relational databases became popular, there was a need for a standard language for data operations. The answer was **SQL (Structured Query Language)**
- Gradually, **SQL** grew into a multipurpose database language with control statements for: creating, modifying and deleting data; data definition (tables, columns); protecting access to database items by working with groups and individual users; data management operations such as backup, block copy and update; and, most importantly, transaction processing.

*The SQL language will be discussed in details in the next topic.*

# Questions and exercises:

1. What are relational languages ?
2. What language does the Relational Model use?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 2. Relational approach in databases

### ❑ Topic 2. Relational languages.

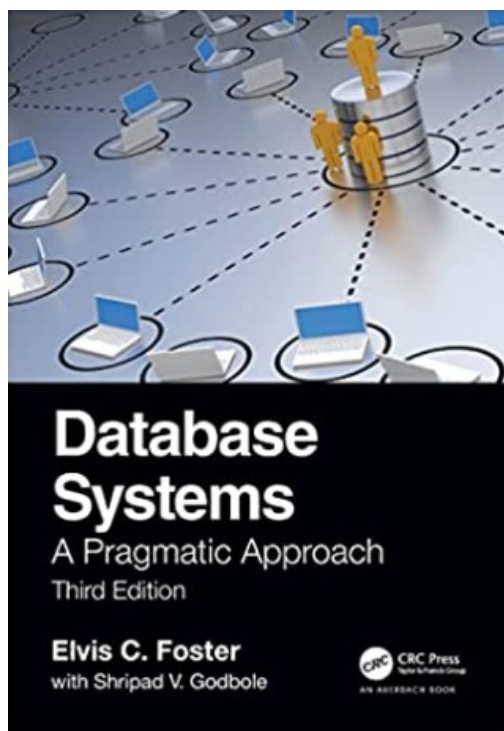
- ❑ Lesson 2. SQL Relational language. Data selection. Built-in functions. Data updating.



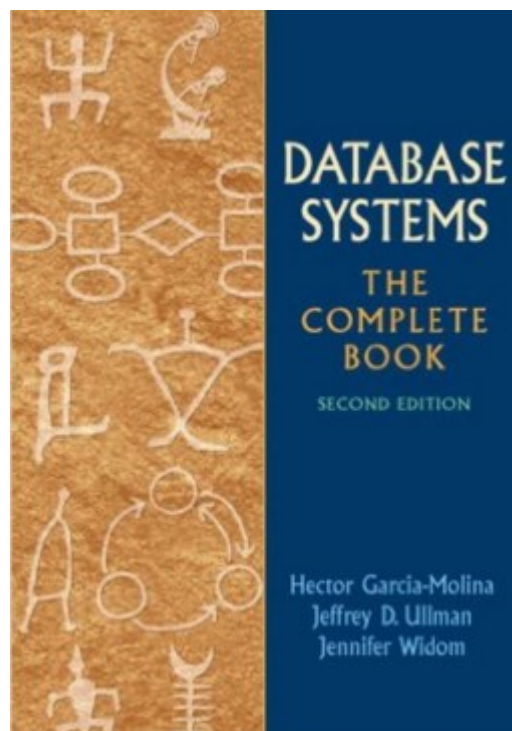
ERASMUS+



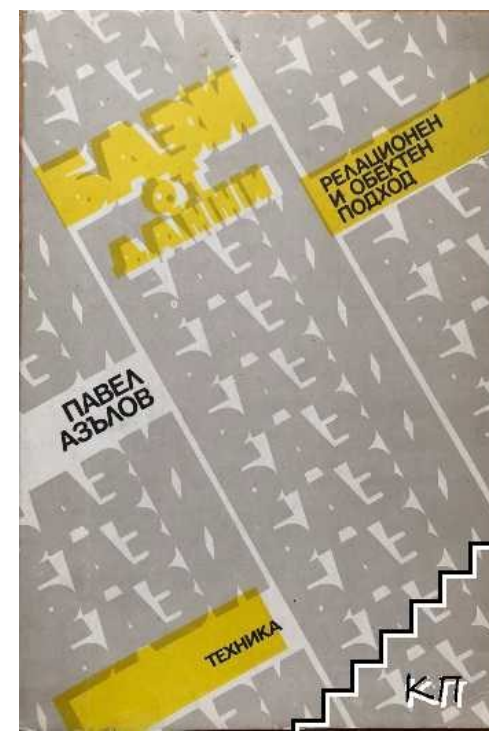
# SQL RELATIONAL LANGUAGE. DATA SELECTION. BUILT-IN FUNCTIONS. DATA UPDATING



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

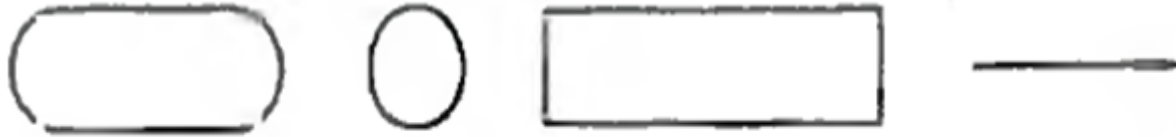
# Introduction

- SQL Language (Structured Query Language), as well as QBE, provides a unified approach for defining and processing data organized in relational databases.
- It was developed in 1974 by IBM associates, one of whom was D. Chamberlin.
- In its initial form, known as SEQUEL, then SEQUEL2, the SQL language was used in various relational systems, such as System R, SQL/DS, DB2, and Oracle.
- SQL operators can be performed in direct mode or embedded in high-level language programs such as PL/1, COBOL, FORTRAN, PASCAL, C, etc.

# Introduction

- The **SQL** language is a non-procedural language due to the fact that its operators specify what is to be obtained, without specifying the way to obtain the desired result.
- The keywords of the language are reserved and cannot be used to denote other objects (tables, columns, etc. ).
- For clarity and brevity, the syntax of the more complex SQL operators is represented graphically with syntax diagrams.
- The basic creative elements of a syntactic diagram are, as usual, the oval, the circle, the rectangle and the arrow:

# Introduction

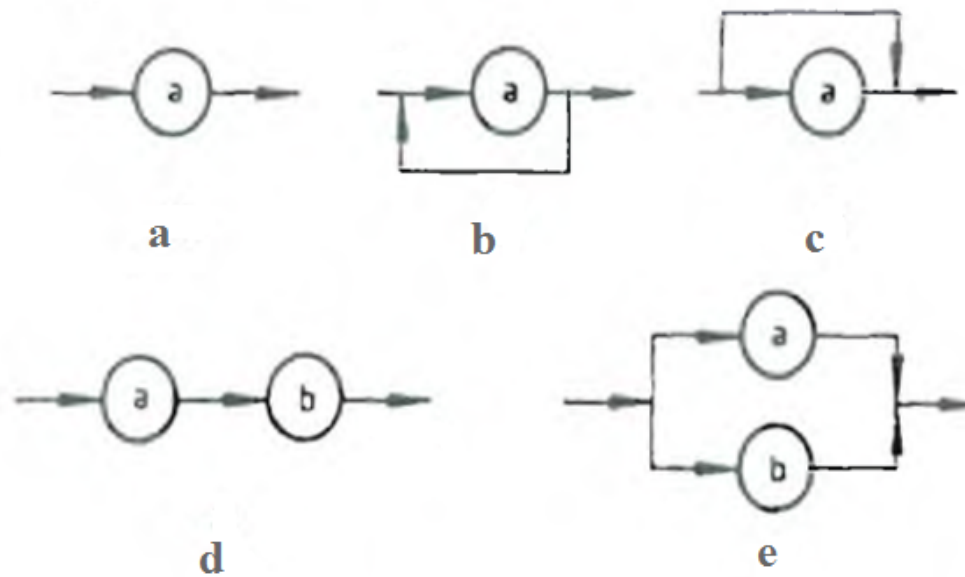


## Basic geometric objects used in syntactic diagrams

- An oval is used to indicate the SQL reserved words. The circle has the same purpose as the oval, but it usually contains words with fewer characters.
- The rectangle is used to indicate nonterminal characters possibly defined in another syntactic diagram.

# Introduction

- The arrow indicates the direction of "movement" (reading) of the syntactic diagram.



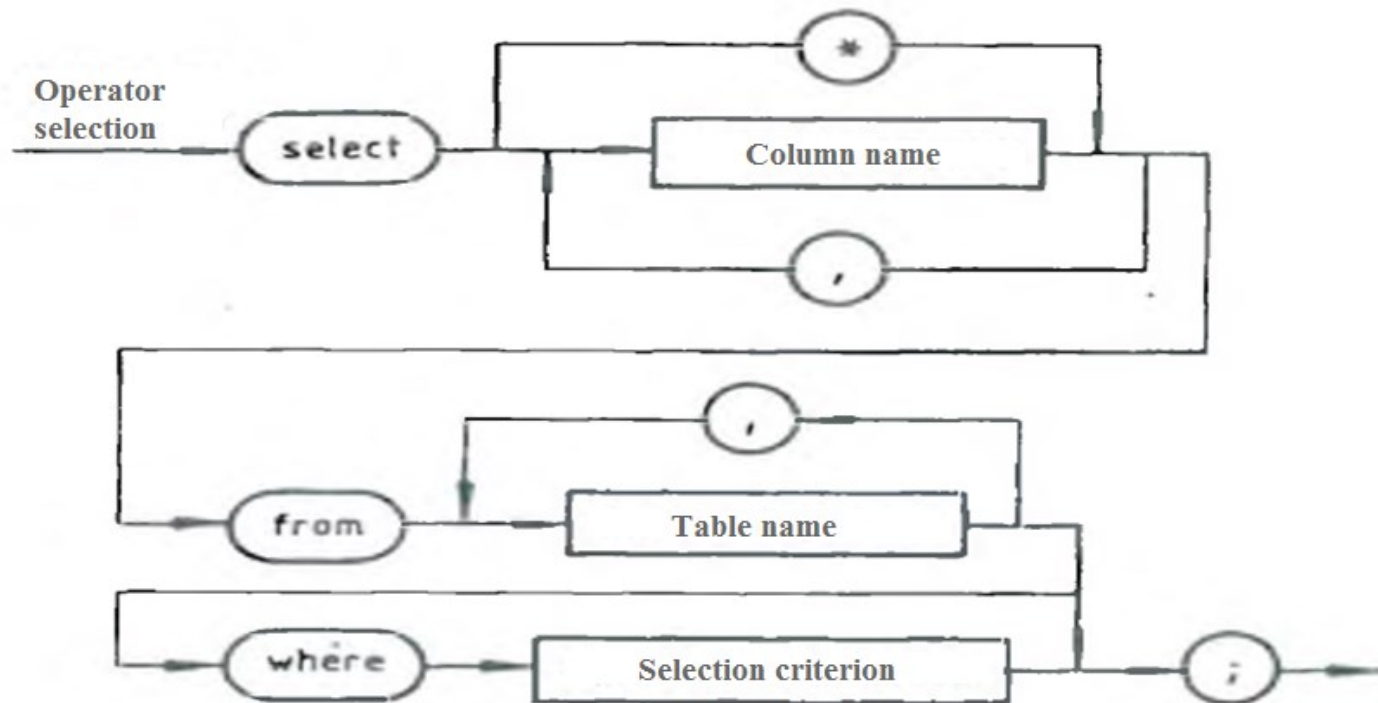
- a* – a character string consisting of only the letter *a* is defined;  
*b* – a character string consisting of one or more characters: *a*, *aa*, *aaa*, *aaaa*, ...;  
*c* – a character string consisting of a single character *a* or is the blank string;  
*d* – a character string *ab* is defined;  
*e* – a character string which is either *a* or *b* is defined

Each syntactic diagram has one output and one input to which the concept being defined is written. "Movement" along the syntactic diagram is according to the direction of the arrows.



# DATA SELECTION

- The selection of data from the DB is done with the select operator. It is most often used in its incomplete form:



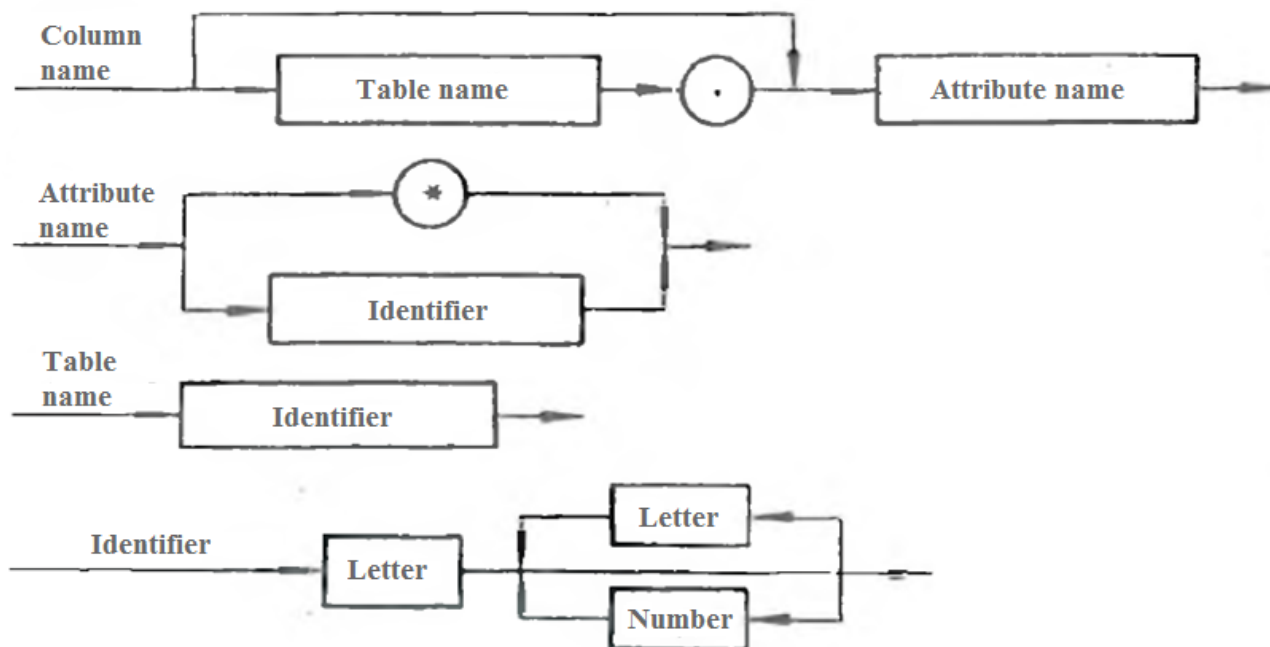
Incomplete format of selection operator

# DATA SELECTION

- The key words in it are:
  - **Select;**
  - **From;**
  - **Where.**
- Identifiers (a character string of letters and numbers beginning with a letter) are used for column and table names.
- As in the other languages, column names are prepended with the name of the corresponding table where necessary.



# DATA SELECTION



Definition of the main objects (names) used in SQL queries

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

## DATA SELECTION

Example                      Get the signature numbers and topics of the books

Solution:                      `select B#, T  
from BOOK;`

Example                      Get the signature numbers and topics of the books,  
published by “Tehnika” Publishing house after 1989

Solution:                      `select B#,T  
from BOOK  
where (E='Tehnika') and (Y>1989)`

# DATA SELECTION

- The selection criterion is a predicate, which in the simpler case may include the operations for relations:

$=, <>, >, <, >=, <=$

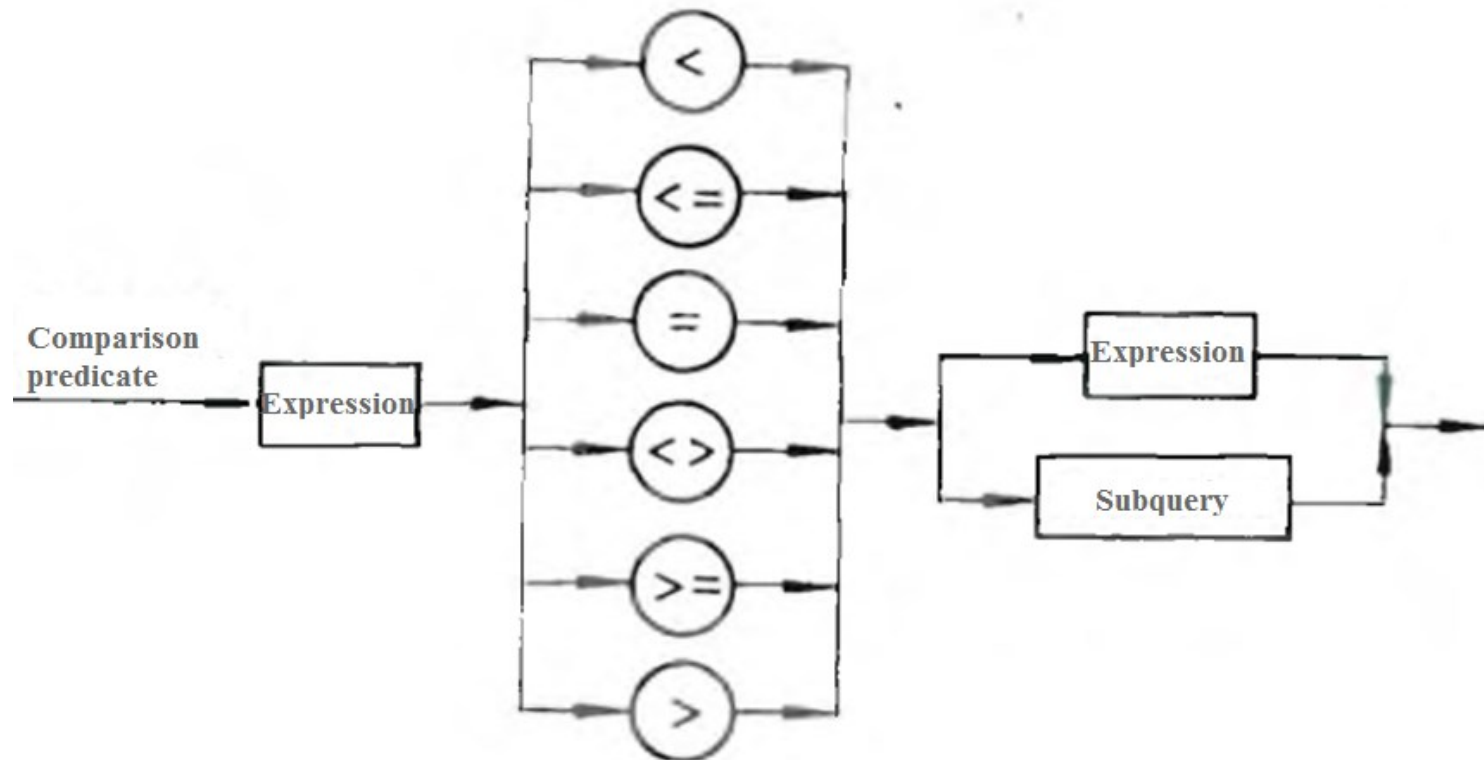
boolean operations:

**and, or, not**

and small brackets.

- The condition that expresses the selection criterion is a predicate for comparing values of expressions:

# DATA SELECTION



Comparison predicate definition

# DATA SELECTION

- As expressions can be used column names, constants (numeric and text), inquiries to built-in functions (count, avr, max, min, sum), and as arithmetic operations - the four basic arithmetic operations (+, -, \*, /).

Example      Obtain complete information about the books  
published in “Tehnika” Publishing house from  
1980 to 1990.

*Solution:*      select \*  
  
from BOOK  
  
where (E='Tehnika') and (Y>1980)  
  
and (Y<=1990)

# DATA SELECTION

**Example** Obtain a name list of all employees and authors of books, published by “Tehnika”.

*Solution:* In the solution of this example union of values of two

comparable attributes BOOK.A and  
EMPLOYEE.N is required.  
This is performed by the union operator.

```
select A
from BOOK
where E='Tehnika'
union
select N
from EMPLOYEE
```

## DATA SELECTION

**Example** Obtain a list of employees that haven't published in "Tehnika" until 1985.

***Solution:*** The answer requires all the employees who have published in "Tehnika" until 1985 to be removed from the multitude i.e. to perform the *minus* operation.

```
select N
from EMPLOYEE
      minus
select A
from BOOK
where (E='Tehnika') and (Y<= 1985)
```



# DATA SELECTION

- The following examples highlight some of the possibilities where the selection criterion is substantially more complex and different from those considered so far.

**Example**            Obtain the names of employees who have taken books from the library.

*Solution:*            To receive an answer to this question tables EMPLOYEE and B\_E are needed. Suppose that the numbers of employees who used books from the library are ( s1, s2, s15). Then the query answer would be as follows:

select EMPLOYEE.N

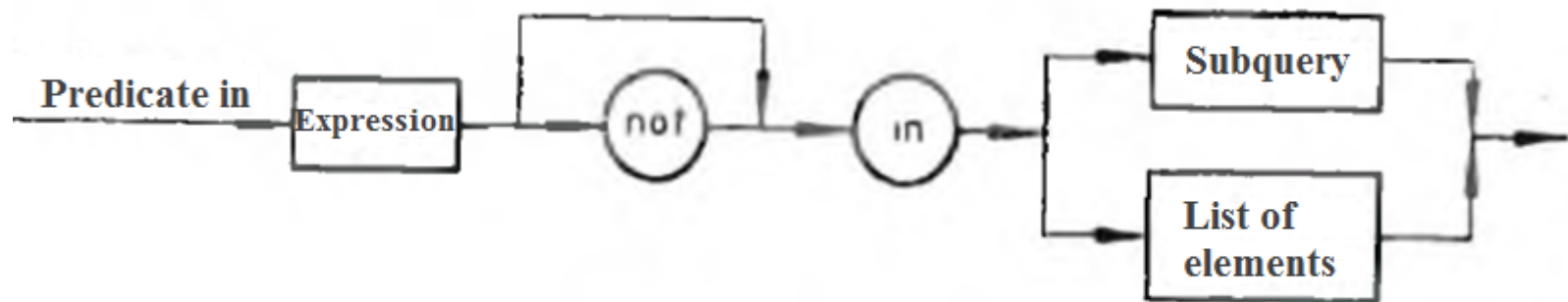
from EMPLOYEE

where EMPLOYEE.S# in (s1, s2, s15)

ERASMUS+

## DATA SELECTION

- The membership predicate **in** checks the membership of the employee number  $S\#$  to the given set.
- The general form of the predicate **in** is given by the syntactic diagram:



**Definition of membership predicate to the set**

# DATA SELECTION

- Since the numbers of employees who used books are generally not known, recording the list of employee numbers (s1,s2,s15) is actually impossible.
- Therefore, this list must be obtained with another query that is a subquery of the above one:

```
select EMPLOYEE.N  
  
from EMPLOYEE  
  
where EMPLOYEE.S# in  
  
(select B_E.S#  
  
from B_E);
```

## DATA SELECTION

**Example**      Obtain the names of employees who are not managers of project p1.

*Solution:*      select EMPLOYEE  
                     from EMPLOYEE  
                     where p1 not in  
                             (select PROJ.P#  
                             from PROJ  
                             where EMPLOYEES#=PROJ.S#)

# Built-in functions

- Like other relational languages, SQL uses several built-in functions.
- These functions can be addressed in the list of items following the word select, as well as in the select criterion.
- Following are some examples illustrating basic use cases of the built-in functions:
  - **Count;**
  - **Sum;**
  - **Min.**

## Built-in functions

Example Obtain the number of all projects.

Solution: select count (P#)  
from PROJ;

Example Obtain the number of the project managers.

Solution: Since one employee can be manager of more than  
one projects only the different numbers of  
employee in table PROJ should be count.

select count (distinct PROJ.P#)  
from PROJ

## Built-in functions

Example

Obtain the total amount of the projects from different subjects (informatics, mathematics, etc.).

Solution:

This is a more general task than this in example 18. In this case the projects should be grouped by research subjects and should be count the total sum for each group of projects. This grouping is indicated by the phrase group by

```
select PROJ.P#, sum(PROJ.C)
from PROJ
group by PROJ.D
```



# DATA UPDATING

- The **update** (change values), **insert** (add rows) and **delete** (remove rows and tables) operators are used to update the database.
- Through concrete examples, the main functions and capabilities of each are outlined.

## Example 24

Double the sum for project number 'p1'

*Solution:*

```
update PROJ  
set C=2*C  
where PROJ.P#='p1'
```

- In the **set** part of the **update** operator, changes can be made to the values of several columns. Assignment operators separated from each other by commas are used for this purpose.

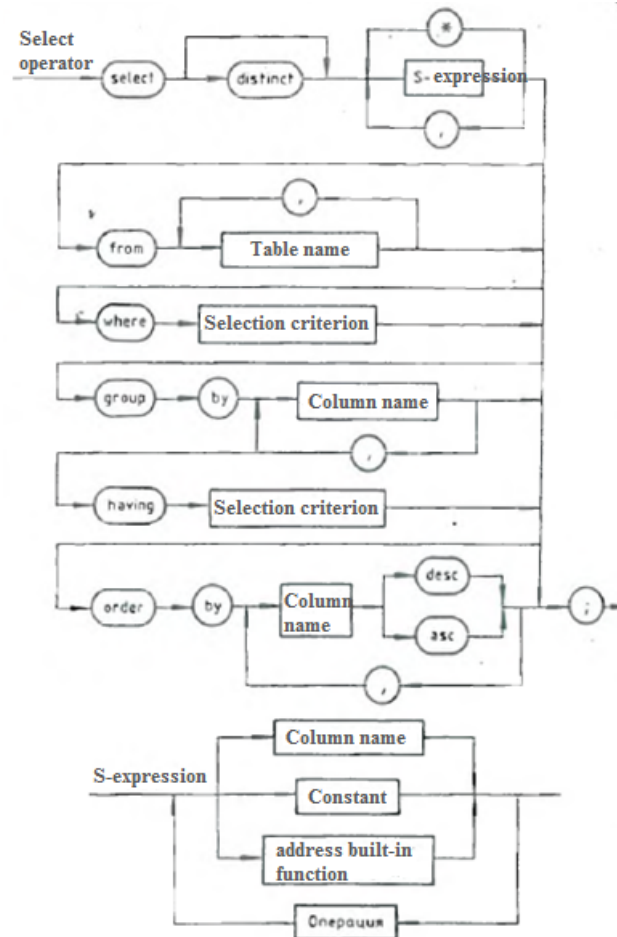
# DATA UPDATING

- The selection criterion can of course be much more complex and thus update the values of a selected subset of table elements

**Example 25**     Add a new row <p7,126,informatics,19 thousand> in table PROJ

*Solution:*     insert  
                  into PROJ  
                  values ('p7',126,'informatics ',19)

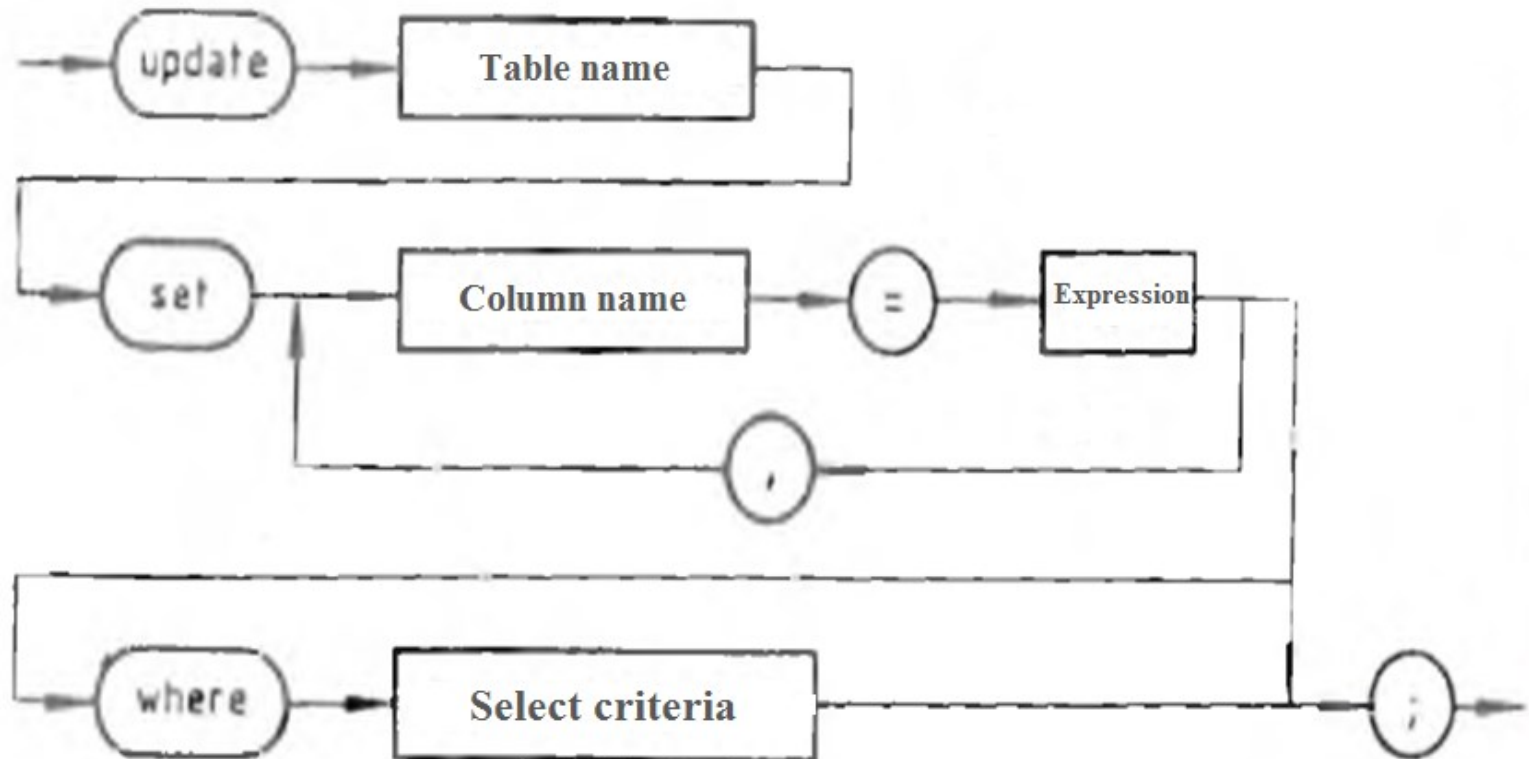
# DATA UPDATING



Full format of the select operator

ERASMUS+

# DATA UPDATING



**General type of change operator**

## DATA UPDATING

- When adding a new row to the table, it is mandatory to specify the values of the key attributes. This is not mandatory for the other attributes, for example:

**insert**

**into PROJ(P#,D,S#)**

**values ('p7','информатика',126)**

Example

All numbers of employees from the PROJ table who are managers of informatics projects to be added to table EMPLOYEE

*Solution:*

insert

into EMPLOYEE (S#)

select PROJ.S#

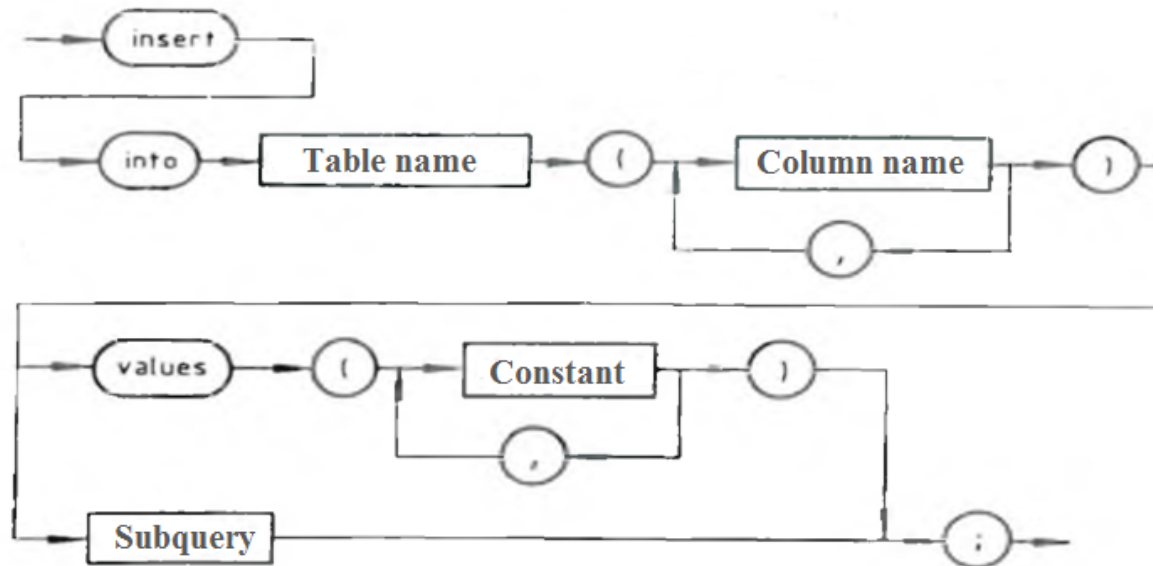
from PROJ

where D='informatics'

ERASMUS+

# DATA UPDATING

- When this operator is performed, the input is done automatically from table PROJ and this is set with the **select** operator.
- The two examples of adding data represent the two varieties of the **insert** operator.



General type of the operator for adding rows to a table

# DATA UPDATING

Example Delete all rows related to books returned in 1970  
from table B\_E

*Solution:* delete  
from B\_E  
where B\_.D2 like %1970;

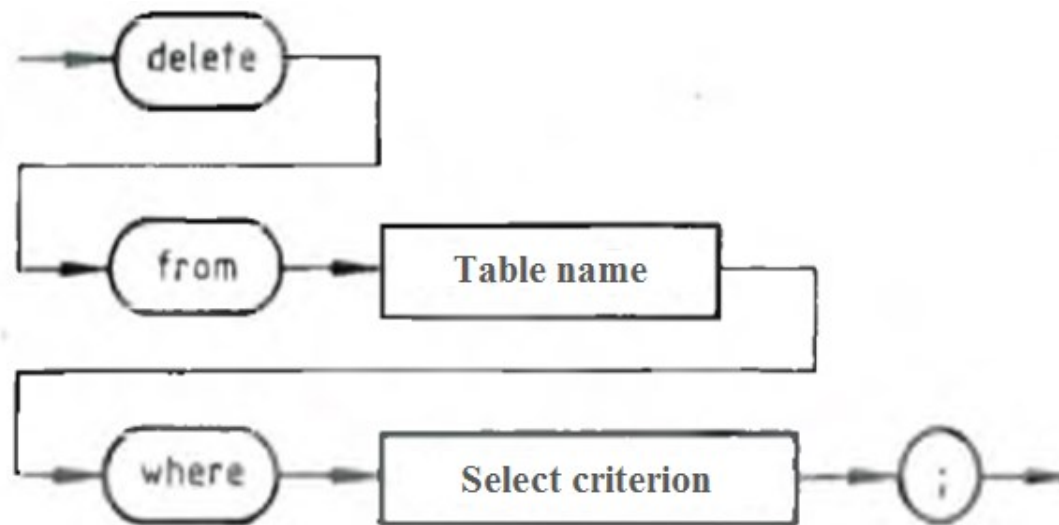
Example Delete all rows from table B\_E

*Solution:* delete  
from B\_E



# DATA UPDATING

- The general appearance of the delete operator is indicated in the figure below:



**Operator for deleting new rows of a table**

# Questions and exercises:

1. What is SQL ?
2. What are Constraints in SQL?
3. What is the correct SQL command to create a database named "library"?
  - ✓ create database library
  - ✓ create new database library
  - ✓ create database library(readers, files, books)
  - ✓ create new database library (readers, files, books)



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 2. Relational approach in databases

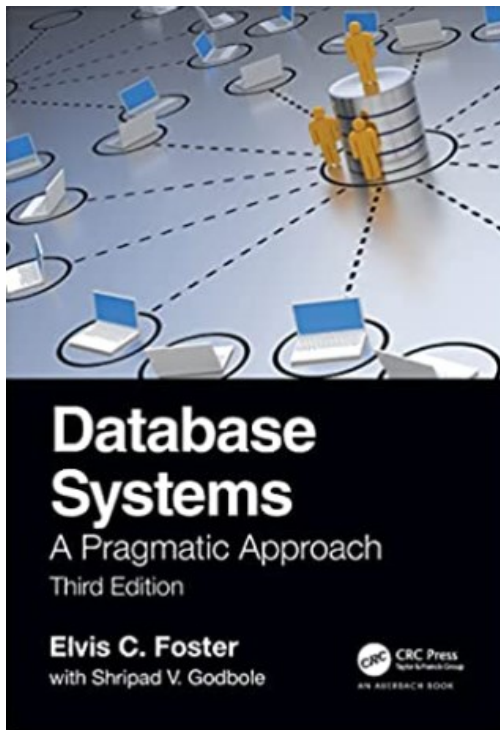
### ❑ Topic 3. Relational systems.

#### ❑ Lesson 1. Basic characteristics and classification of relational systems

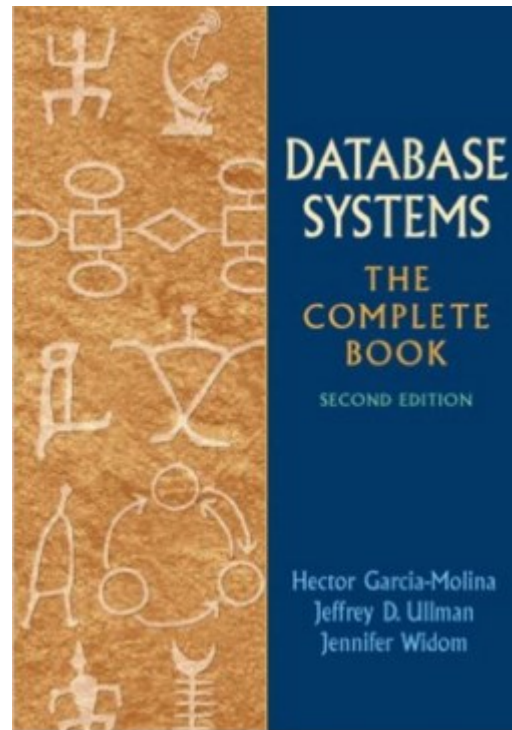


ERASMUS+

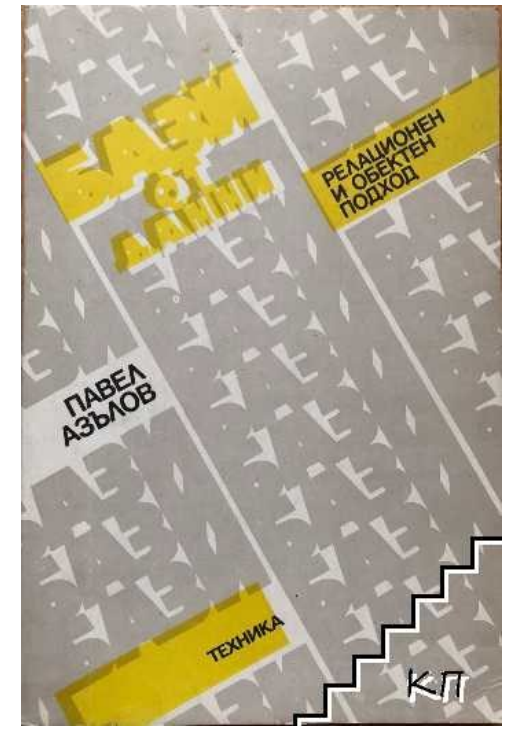
# BASIC CHARACTERISTICS AND CLASSIFICATION OF RELATIONAL SYSTEMS.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# Introduction

- Modern DBMSs emerged as a necessary programming tool to increase the productivity of programmers in designing and developing information processing systems, and to increase the role of the end user in creating, modifying, accompanying, and operating specific programming systems.
- In the opinion of many specialists, the requirements of any DBMS are most satisfied by DBMSs using the relational data model.



# Basic characteristics of relational systems.

- These systems are called **relational database management systems**, or **RDBMS** for short.
- The main advantages of RDBMS built on the relational data model are:
  - ***Simplicity***. It is expressed in the homogeneity of the logical representation of the data and the relationships between them.
  - ***Data independence***. Relational schemas and relational languages are independent of data representation methods and data access methods.
  - ***Ability to optimize queries***.



# Basic characteristics of relational systems.

- *High-level interface.* Through non-procedural relational languages, the end user has the ability to describe their queries to the DB.
- *Ability for effective physical presenting and searching*
- *Work with virtual objects.* Based on defined relations, the user has the ability to define virtual, physically non-existent relations.
- *Distributed processing capability.*
- *Strong theoretical basis.* It is based on the mathematical concept of n-membered relationality, set theory and first-order predicate calculus.

# Relational systems classification

- When classifying relational systems, the lowest level is the so-called *minimal relational systems*.
- These are systems that support only relational structures, do not support integrity constraints, and their corresponding DML (Data manipulation language) is not relationally complete.
- At the next level are *relationally complete systems*, which, except for relational structures, support relationally complete DML but without null value processing.
- Highest in the hierarchy are *fully relational systems*. They support relational structures, their corresponding DML is relationally complete, and they support integrity constraints

# Relational systems classification

- Fully relational systems are described in detail by E. Codd. The requirements defining when a system is fully relational are briefly listed below:
  1. Each piece of information in the RDBMS shall be presented in a tabular format. (In addition to the data in the DB and the relationships between them, tabular descriptions of the data, integrity constraints, etc. )
  2. Each data element in the DB is accessible by a table name, a primary key value, and a column name.
  3. Ability to process null (non-key) values.

# Relational systems classification

4. The system catalog is a set of tables that are processed with the relational language available in the system.
5. Relational language should include:
  - data description language, including the definition of virtual tables;
  - data processing language;
  - language for defining integrity constraints.
6. Ability to update virtual tables.
7. Ability to update entire tables.

# Relational systems classification

8. Physical data independence.

9. Logical independence.

10. "Independence" of integrity:

- integrity constraints are defined by means of the relational language, not within the user program;
- the set of integrity constraints must include at least the two main structural integrity constraints.

11. Ability for centralized and distributed processing.

# Relational systems classification

12. If a low-level language is also present in the system, it must not allow ways to violate the integrity constraints introduced by the high-level relational language.

*As you can see, the requirements for relational systems are extremely serious. This fully explains the fact that on the software market it is rare to find systems that satisfy all twelve requirements, but that is why developers strive to make their systems as close as possible to fully relational systems.*

# An overview of some basic relational systems

- Several relational systems are selected here, which reflect the relational approach in different ways with their variety of linguistic tools.

## INGRES SYSTEM

- INGRES (INteractive Graphics and REtrieval System) is a relational system developed by the University of California, Berkeley.
- The main achievements in INGRES are the high data independence and the availability of a non-procedural language for description, search, update data and maintain integrity constraints. In this sense, it comes too close to fully relational systems.



# An overview of some basic relational systems

- Four interfaces are supported in INGRES:
  - QUEL - a relationally complete query language based on relational calculus with n-tuples variables;
  - EQUQL - an interface through which QUEL operators can be included in programs of C language;
  - CUPID - a non-procedural interface through pictures with search and update capabilities;
  - GEO-QUEL - an interface which is used to display relevant data from the database in the form of geographical maps.

# An overview of some basic relational systems

- The selective possibilities of the QUEL language are expressed by an operator similar to the get operator of the Alpha language, but without quantifiers:  

```
range of t1 is R1
range of t2 is R2
...
retrieve (t1.A, t2.B, ...) where C
```
- Here t1 and t2 are names of variable n-tuples whose modification domains are respectively R1 and R2, A and B are attribute names in R1 and R2, and C is a condition on which the selection of data from the DB is performed.

*QUEL allows the definition of virtual tables and indexes, as well as access by hashing*

# An Overview of some basic relational systems

## ORACLE SYSTEM

- ORACLE is a fully relational system and was developed by the company Relational Software Inc. Its initial version is from 1978. The language used in the system is SQL and it can be worked with either directly or through programs in PASCAL, C, PL/1, COBOL, FORTRAN, which include SQL calls.
- ORACLE is used for microcomputers and for large computers running various operating systems.
- A report generator is created to the system with options for:



# An Overview of some basic relational systems

- development of interactive applications;
  - outputting reports;
  - editing texts
- ORACLE has advanced possibilities for defining and processing virtual tables, as well as ensuring data integrity in the DB.

## PASCAL/R SYSTEM

- PASCAL/R is a relational system that is a relational extension of the PASCAL algorithmic language. It was developed at the University of Hamburg and the initial version dates back to 1978. PASCAL/R system supports three interfaces:

# An Overview of some basic relational systems

- through PASCAL/R programs;
  - interactive interface;
  - privileged interface for the DB administrator.
- DB description is done in the types definition section of a PASCAL/R program.
  - Data selection is done using relational expressions of the form:

```
some t in R : p(t);  
each t in R : p(t);
```

where  $t$  is an  $n$ -tuple name,  $R$  is a relation name, and  $p(i)$  is a predicate.

# An Overview of some basic relational systems

## Example

```
var ORDERLIST : file of Item;

...
with MYDATABASE do
  begin
    ...
    for each P in Parts : (P.Colour = blue) and
      some SP in Shipments : (SP.PNR = P.PNR) do
      begin
        ORDERLIST^ := P;
        put(ORDERLIST)
      end
    end;
  end;
```

- Boolean relational expressions can be used in loop operators and in the conditional operator. The system does not support virtual tables.

# An Overview of some basic relational systems

## QUERY-BY-EXAMPLE SYSTEM

- QUERY-BY-EXAMPLE (QBE) is a relational system developed in 1978 in the laboratory of IBM, USA. The extended version of this system is called OPE (Office Procedures By Example).
- The language used in the system is QBE. In addition to directly, the user can interact with the system via programs compiled in PL/1 or APL, in which QBE addressing is allowed. The system also allows the following possibilities:
  - definition of integrity constraints;



# An Overview of some basic relational systems

- generation of a new DB and reorganization of an existing DB;
  - generator of outputs;
  - creating indexes, inverting columns.
- The system supports four types of system tables in the catalog named TABLE, DOMAIN, PROGRAM, and AUTHORITY. The last two tables contain program or DB query names and access rights for individual users.

# An Overview of some basic relational systems

## SYSTEM R SYSTEM

- SYSTEM R is a completely relational system developed in 1975 by IBM Laboratory, USA. The language used in the system is SQL.
- The user works with the system in dialog mode, using SQL commands or through PL/1 or COBOL programs where he inserts SQL commands.

# An Overview of some basic relational systems

## MySQL SYSTEM

- MySQL is an open source relational database management system (RDBMS) that uses SQL, the most popular language for entering, accessing, and performing other data processes in a database.
- Because it's an open source, anyone can download MySQL and add to it, depending on the general public rules.
- MySQL is mainly known for its speed, security and flexibility. However, it is well-known that it works best when it manages content and does not perform transactions.

# An Overview of some basic relational systems

## Microsoft Access SYSTEM

Microsoft Access is a completely functional relational database management system. It provides all the possibilities for data processing and control:

1. Data definition and storage. Microsoft Access has flexible options for defining different types of data (text, numbers, dates, currency, pictures, sounds, documents, spreadsheets, Internet connections).

The user can define the way the data is stored and displayed on the screen or printer. Of particular importance is the ability to define simple and sophisticated data validity rules which ensure entering the correct into the database.

# An Overview of some basic relational systems

2. Data processing. Microsoft Access uses the powerful SQL relational data model language. To obtain the required result information, it has a powerful means of defining queries that are created without the need for the user to have knowledge of SQL. Queries can be created incorporating data from many tables.

3. Data control. Microsoft Access can be used on a single workstation or in client/server mode on a network. For complete and efficient data sharing with other users, Access has excellent security and data integration capabilities. You can define which users or groups of users have access to the objects and the level of access - to read the data or to be able to edit and update it.



# An Overview of some basic relational systems

## Microsoft SQL Server SYSTEM

- Microsoft SQL Server is an overall relational database management and data analytics system that is used to create a variety of e-commerce solutions, business applications, and big data storage.
- Like any Microsoft product, there are many different versions of Microsoft SQL Server. They are designed for a variety of audiences and workloads, ranging from small single-machine applications to huge Internet-based applications that many users are working on simultaneously.
- Microsoft SQL Server is designed to manage large server-based DBs as opposed to MS Access which is desktop-based and is not designed to manage large corporate DBs.

# Questions and exercises:

1. What is RDBMS? How is it different from DBMS?
2. What are the basic characteristics of RDBMS?





# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 2. Relational approach in databases

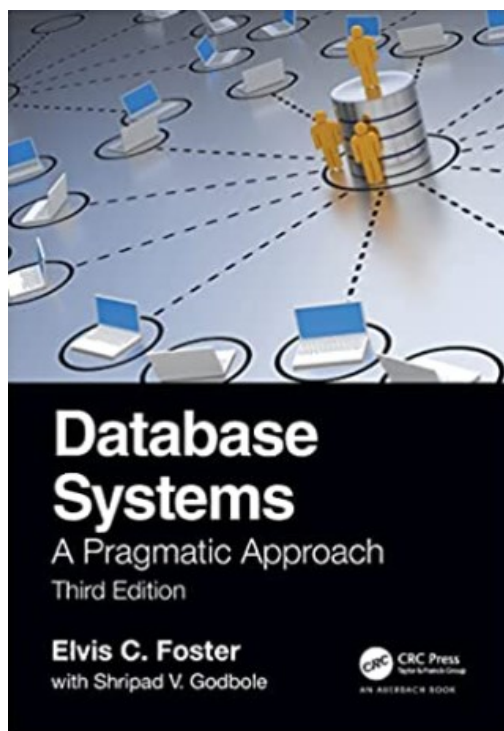
### ❑ Topic 3. Relational systems.

#### ❑ Lesson 2. Relational schema analysis. Functional dependencies

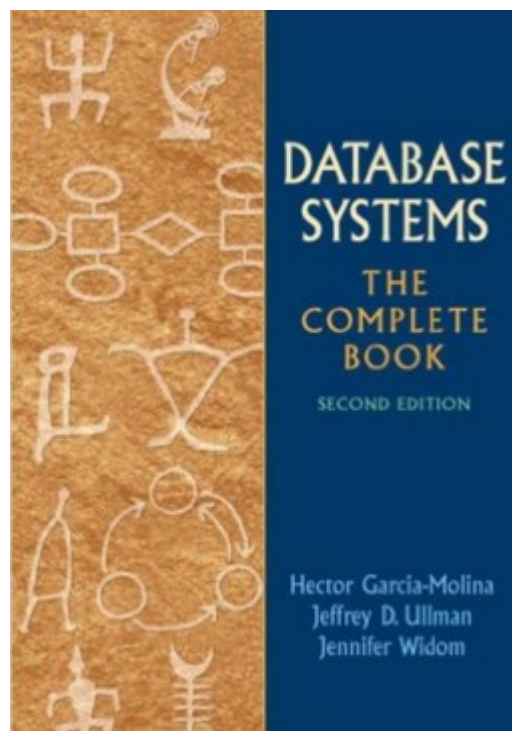


ERASMUS+

# RELATIONAL SCHEMA ANALYSIS. FUNCTIONAL DEPENDENCIES.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# Introduction

- A major problem in building models of a subject area is the definition of the entities (relations, tables) that are chosen for classes of objects in the subject area and the properties (attributes, branches) that characterize them.
- The complexity of the problem is a consequence of the ambiguity of its solution.
- In such case the following questions may arise: *when a relational schema representing an object class is good; when two schemes are equivalent or which of two schemes is the better one.*
- Each of these questions actually leads to the need of a formal tool for analyzing relational schemas.

# Introduction

- Before presenting methods for the analysis and evaluation of relational schemes, an example is presented to illustrate various deficiencies of one particular relational scheme.
- In literature, these deficiencies are known as *redundancy, updating, addition, and deleting anomalies*.
- A relation scheme is presented:

**LIBRARY**      (   **Lib#, Address, Book, Number**   )

where each library is presented by it's number (**Lib#**) and address (**Address**) and each book is presented by it's name (**Book**) and it's number of copies (**Number**) that are available in the library.

# Introduction

- It is natural to assume that:
  - each library is uniquely defined by its number;
  - one book can be stored in several libraries.
- As a consequence, the following conclusions can be made:
  1. For each book from the same library there will be a repeat of the library address.
  2. It follows from point (1) that when the library address changes, it will be necessary to update as many rows of the relation as there are books stored in it.



# Introduction

This operation does not exclude the possibility that some of the rows may remain non-updated, where **inconsistencies** in the DB data will occur.

3. The address of a newly created library cannot be entered into the DB until at least one book that will be stored in that library is specified. The reason for this is that the key attributes in this relational schema are the Lib# and Book attributes, and therefore they must have well-defined values for each element of the relation
4. If, due to renovations to a library, the books that were stored in that library are moved to another book repository or library, all the rows in the relation pertaining to that library have to be removed from the relation.



# Introduction

As a consequence, the address of the library will also be "lost", even though the library exists and its address remains the same.

The above deficiencies, called redundancy anomaly (1), update anomaly (2), add anomaly (3) and remove anomaly (4), show that the relational scheme LIBRARY is not "good".

The methods that lead to its "improvement" require first to present methods for analyzing the dependencies that exist explicitly or implicitly between the attributes of a relational schema.

# Introduction

- The methods that lead to its "**improvement**" require first to present methods for analyzing the dependencies that exist explicitly or implicitly between the attributes of a relational schema.

Most often, two types of dependencies are considered - functional and multivalued.

Functional Dependency is a constraint that determines the relation of one attribute to another attribute in a Relation Database Management System.

# FUNCTIONAL DEPENDENCIES

- Let  $A_1, A_2, \dots, A_n$  be a list of  $n$  attributes,  $n \geq 1$ , which we will denote by  $A_1 A_2 \dots A_n$  (without commas between the attribute names) or with only one name  $A$ ,  $A = A_1, \dots, A_n$ .
- It is assumed that the list  $A$  is unordered and that there are no repeated elements, i. e.  $A$  is a set of attribute names. By  $P(A)$  we denote the set of all subsets of  $A$ .
- It is known that if  $A$  has  $n$  elements, i. e.  $|A| = n$ , then the number of subsets of  $A$  is  $2^n$ , i. e.  $|P(A)| = 2^n$ .
- Let  $R(A)$  be a relational scheme and  $X, Y \in P(A)$ .

# FUNCTIONAL DEPENDENCIES

- An attribute  $X$  is said to functionally determine an attribute  $Y$  if for any relation  $r$  with relational schema  $R(A)$  given any two rows  $i_1, i_2 \in r$  for which  $i_1.X = i_2.X$ , it follows that  $i_1.Y = i_2.Y$ , when rows are equal in the  $X$  attribute, they are equal in the  $Y$  attribute.
- It should be noted that  $X$  and  $Y$  can be both *simple attributes and composite attributes*, i. e. they can consist of several simple attributes.
- When  $X$  functionally determines  $Y$ , it is also said that  $Y$  is functionally dependent on  $X$  or that there is a functional dependence of  $X$  in  $Y$ , using the notation  $R.X \rightarrow R.Y$ .

# FUNCTIONAL DEPENDENCIES

- If there is no ambiguity, specifying the relational schema is optional.
- It can be noticed that in fact the functional dependence between two attributes can also be considered as a completeness constraint on the relational schema  $R(A)$  of the form  $X \rightarrow Y$ , where  $X, Y \in P(A)$ .
- This restriction guarantees that for any relation  $r$ ,  $r:R(A)$ , any value of the attribute  $X$  uniquely determines the value of the attribute  $Y$ .
- For the formal expression of a functional dependence, operations from relational algebra can also be used.

# FUNCTIONAL DEPENDENCIES

- Often the set of functional dependencies is denoted by  $F$ :

$$F = \{(X, Y) : X \twoheadrightarrow Y, X \in \mathcal{P}(A), Y \in \mathcal{P}(A)\},$$

i. e.  $F$  is a binary relation.

- It is also customary to say that if  $X \twoheadrightarrow Y$  belongs to  $F$ , then  $X \twoheadrightarrow Y$  is an  $F$ -dependence.
- If  $X \twoheadrightarrow Y$  is an  $F$ -dependence that is satisfied by the relation  $r$ , in short we will write this with  $r(X \twoheadrightarrow Y) = \text{true}$ , and in case  $X \twoheadrightarrow Y$  is not an  $F$ -dependence for  $r$ , we will write  $r(X \twoheadrightarrow Y) = \text{false}$ .

# FUNCTIONAL DEPENDENCIES

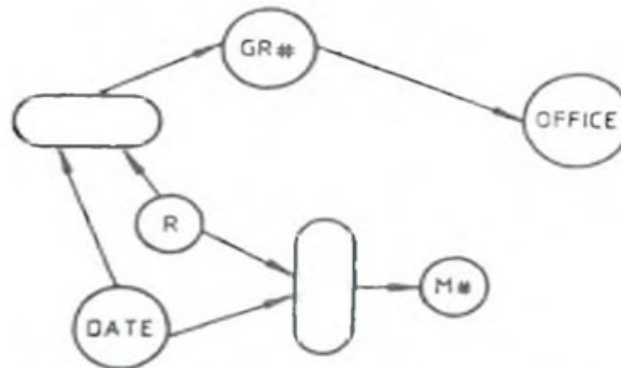
- Since the set of F-dependencies is a binary relation, a convenient and visual way of representing it is by a graph  $G = (A, F)$ , where  $A = A = \{A_1, A_2, \dots, A_n\}$  is a set of attributes, and  $F$  is a set of F-dependencies of the form  $X \dashv\dashv> A, X \in P(A)$ .
- On each attribute  $A_i \in A$  we match a vertex of the graph.
- When  $X$  is a simple attribute, then we map an arc to the F-dependence  $X \dashv\dashv> A_i$ .
- However, there are cases where  $X$  is a composite attribute. In such case, vertices of the second kind corresponding to the constituent attributes are entered.



# FUNCTIONAL DEPENDENCIES

- If  $X = A_{i1}, A_{i2}, \dots, A_{ik}$ , then the arcs  $(A_{ij}, X), j = 1, 2, \dots, k$  are built first and then the arc  $(X, A_i)$ .

## F-dependencies of the relational scheme EXCURSION



EXCURSION (OFFICE#, R, GR#, M#, DATE)

### Attributes

OFFICE# - Travel agency (number);  
R - Excursion group guide  
GR# - Excursion group number  
M# - Route number  
DATE - Departure date

### F-dependencies

(R, DATE)  $\longrightarrow$  GR#  
(R, DATE)  $\longrightarrow$  M#  
GR#  $\longrightarrow$  OFFICE#

F-dependencies graph sample

ERASMUS+

# AXIOMATICS OF FUNCTIONAL DEPENDENCIES

- We consider the 3-member relation  $r$ :

$r$	A	B	C
	a	b	c
	a	b1	c
	a1	b	c1

- From the first and second rows of the tabular representation of  $r$  it could be thought that  $A \twoheadrightarrow C$ .
- However, this conclusion would not be true without knowing all the states of the relation  $r$ , since F-dependence is a property of the relational scheme, not of a particular relation.

# AXIOMATICS OF FUNCTIONAL DEPENDENCIES

- In fact, each  $F$ -dependency complements the semantics of the modeled class of objects since it is invariant with respect to the current information content of the DB.
- The dependencies that it is possible to define within a relational scheme, even though finite in number, can sometimes be too many.
- It is therefore natural to ask, how on a given set of functional dependencies  $F$  can all functional dependencies be obtained?

# AXIOMATICS OF FUNCTIONAL DEPENDENCIES

- An F-dependence  $X \twoheadrightarrow Y$  is called a logical consequence of the set of functional dependencies  $F$  if for any relation  $r$  satisfying the F-dependences  $F$ , i. e.  $r(F)=true$ , it follows that  $r(X \twoheadrightarrow Y)=true$ .
- In short, we will denote this by  $F:= X \twoheadrightarrow Y$ .
- Armstrong shows that by using the so-called inference rules or axioms for a given set of F-dependencies, new F-dependencies are obtained, and all F-dependencies can be obtained.

# AXIOMATICS OF FUNCTIONAL DEPENDENCIES

- In the formulation of the inference axioms given below it is assumed that:  $R(A)$  is a relational scheme;  $F$  is the set of F-dependencies in  $A$ ;  $X, Y, Z$  and  $W$  are lists of attributes of  $A$ ;  $r$  is an arbitrary relation with scheme  $R(A)$ .

*F1. Reflectiveness*

If  $Y \subseteq X$ , then  $F \models X \twoheadrightarrow Y$ .

*F2. Augmentation*

If  $X \twoheadrightarrow Y$ , then  $F \models XZ \twoheadrightarrow YZ$ .

*F3. Transitivity*

If  $X \twoheadrightarrow Y$  and  $Y \twoheadrightarrow Z$ , then  $F \models X \twoheadrightarrow Z$ .

# AXIOMATICS OF FUNCTIONAL DEPENDENCIES

- The proof of these rules follows immediately from the definition of  $F$ -dependence.
- For example, the proof of the first is: if there exist elements  $i_1$  and  $i_2$  of  $r$  such that  $i_1.X=i_2.X$ , since  $Y \subseteq X$  it follows that  $i_1.Y=i_2.Y$ , i. e.  $F1$  is valid.
- Similarly,  $F2$  and  $F3$  can be proved.
- Based on  $F1$ ,  $F2$  and  $F3$  other inference rules can be proved.

# AXIOMATICS OF FUNCTIONAL DEPENDENCIES

## *F4. Unification*

If  $X \multimap Y$  and  $X \multimap Z$ , then  $X \multimap YZ$ .

Really from  $X \multimap Y$  and from F2 follows that  $X \multimap XY$ , and from  $X \multimap Z$  and from F2 follows that  $XY \multimap ZY$ . Then according to F3  $X \multimap YZ$ .

## *F5. Pseudo transitivity*

If  $X \multimap Y$ ,  $WY \multimap Z$ , then  $XW \multimap Z$ .

Really, from  $X \multimap Y$  and from F2 follows that  $XW \multimap YW$ .  
Then from F3 follows that  $X \multimap Z$ .

## *F6. Decomposition*

If  $X \multimap Y$  and  $Z \subseteq Y$ , then  $X \multimap Z$ .

Really, from  $Z \subseteq Y$  and F1 follows that  $Y \multimap Z$   
and from F3 finally follows F6



# AXIOMATICS OF FUNCTIONAL DEPENDENCIES

- From the axioms  $F_4$  and  $F_6$  the consequence of unification and decomposition can be formulated:

$X \text{ --- } > Y_1, Y_2, \dots, Y_k$ , if and only if

$X \text{ --- } > Y_i, \exists a \ i = 1, 2, \dots, n$

# CLOSURES AND COVERS

- The set of  $F$ -dependencies that are logical consequences of  $F$  is called  $F$  **closure** and is denoted by  $F^+$ .
- Finding the  $F^+$  closure of a set of functional dependencies  $F$  in general case is not easy to do.
- That is mainly because usually the number of closure attributes is too big.

# CLOSURES AND COVERS

Example: A relational scheme  $R(A,B,C)$  with functional dependencies  $F=\{A \twoheadrightarrow B, B \twoheadrightarrow C\}$  is given. Define the  $F^+$  closure as F-dependencies if  $\phi \twoheadrightarrow X$  or  $X \twoheadrightarrow \phi$  are not considered.

- |                                 |                                  |
|---------------------------------|----------------------------------|
| 1. $A \twoheadrightarrow A$     | 19. $AC \twoheadrightarrow A$    |
| 2. $A \twoheadrightarrow B$     | 20. $AC \twoheadrightarrow B$    |
| 3. $A \twoheadrightarrow C$     | 21. $AC \twoheadrightarrow C$    |
| 4. $A \twoheadrightarrow AB$    | 22. $AC \twoheadrightarrow AB$   |
| 5. $A \twoheadrightarrow AC$    | 23. $AC \twoheadrightarrow AC$   |
| 6. $A \twoheadrightarrow BC$    | 24. $AC \twoheadrightarrow BC$   |
| 7. $A \twoheadrightarrow ABC$   | 25. $AC \twoheadrightarrow ABC$  |
| 8. $B \twoheadrightarrow B$     | 26. $BC \twoheadrightarrow B$    |
| 9. $B \twoheadrightarrow C$     | 27. $BC \twoheadrightarrow C$    |
| 10. $B \twoheadrightarrow BC$   | 28. $BC \twoheadrightarrow BC$   |
| 11. $C \twoheadrightarrow C$    | 29. $ABC \twoheadrightarrow A$   |
| 12. $AB \twoheadrightarrow A$   | 30. $ABC \twoheadrightarrow B$   |
| 13. $AB \twoheadrightarrow B$   | 31. $ABC \twoheadrightarrow C$   |
| 14. $AB \twoheadrightarrow C$   | 32. $ABC \twoheadrightarrow AB$  |
| 15. $AB \twoheadrightarrow AB$  | 33. $ABC \twoheadrightarrow AC$  |
| 16. $AB \twoheadrightarrow AC$  | 34. $ABC \twoheadrightarrow BC$  |
| 17. $AB \twoheadrightarrow BC$  | 35. $ABC \twoheadrightarrow ABC$ |
| 18. $AB \twoheadrightarrow ABC$ |                                  |

# CLOSURES AND COVERS

- Finding all  $F$ -dependencies is not always necessary. Sometimes it is enough just to know whether or not an  $F$ -dependency belongs to the closure.
- For example, if considering the relational schema  $R(A1, A2, A3, A4, A5, A6, A7)$  with  $F$ -dependencies  $F\{A1 \twoheadrightarrow A4, A1A2 \twoheadrightarrow A4A5, A3A5 \twoheadrightarrow A6, A5 \twoheadrightarrow A7\}$ , then it is obvious that checking whether  $A1A2 \twoheadrightarrow A5A7$  is an element of  $F^+$  does not require first determining  $F^+$ .

# CLOSURES AND COVERS

- The use of direct methods requires introducing the notion of closure attribute.
- Let  $F$  be the set of F-dependencies in  $R(A)$  and  $X \in P(A)$ . The set  $X^+$  is called a closure of  $X$  if for any  $Y$ ,  $Y \in P(A)$  where  $X \twoheadrightarrow Y$  is an F-dependence, it follows that  $Y \subseteq X^+$ .
- In other words, the closure  $X^+$  of an attribute  $X$  (possibly composite) is the maximal attribute for which  $X \twoheadrightarrow X^+$  is from  $F^+$ .

# CLOSURES AND COVERS

- A set of F-dependencies  $F$  is called a redundancy-free cover if no proper subset of  $F$  is equivalent to  $F$ .
- It means that if  $F$  is a cover without redundancy, then for every  $F'$ ,  $F' \subset F$ , we have  $(F')^+ \neq F^+$ .
- It is not difficult to check that, for example, the set  $G = \{A \multimap B, B \multimap C, AB \multimap C, AC \multimap B\}$  is a *redundancy cover* of the set of F-dependencies  $F = \{A \multimap B, B \multimap C\}$ , since  $F \subset G$  and  $F^+ = G^+$ .
- It is interesting to note that there are cases where a set of F-dependencies can have several covers without redundancy.

# Questions and exercises:

1. What is functional dependency?
2. What are rules of functional dependencies ?
3. Which is types of Functional Dependencies in DBMS ?





Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

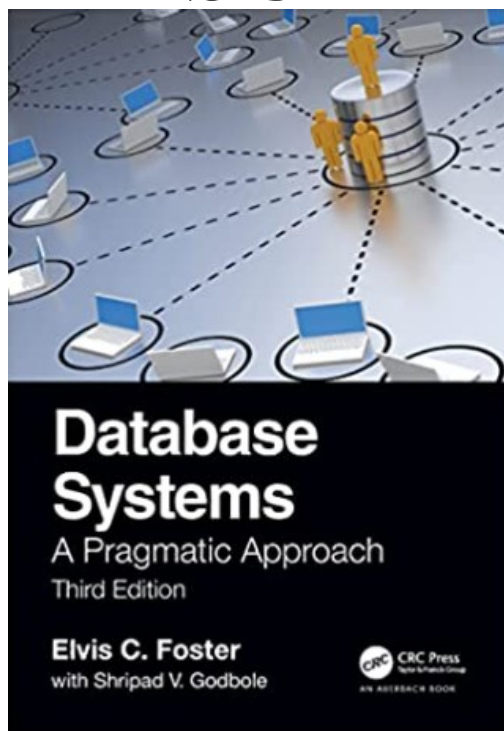
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 2. Relational approach in databases**
- ❑ **Topic 4. Development of a sample database. Normalization.**
- ❑ **Lesson 1. Normalization of relational schemas. Normal forms.**

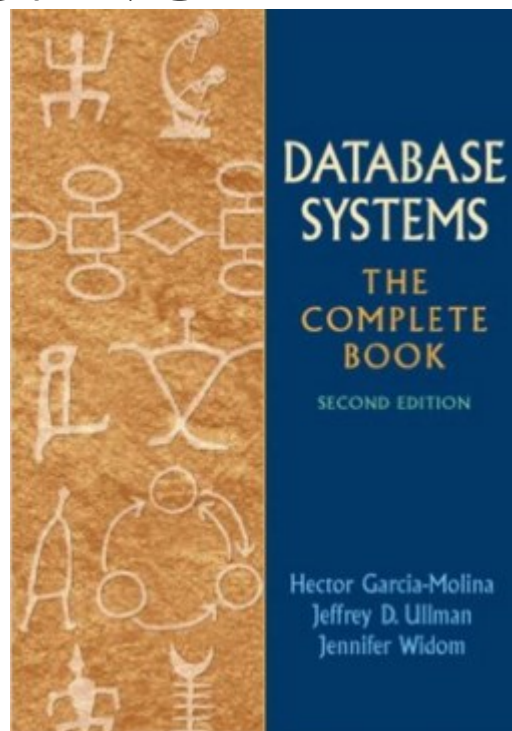


ERASMUS+

# NORMALIZATION OF RELATIONAL SCHEMAS. NORMAL FORMS.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# Normalization of relational schemas. Normal forms.

- Working with any relational schema  $R(A)$  involves using keys that are a subset of the set of attributes  $A$ .
- The values of these attributes in the individual rows of each relation  $r$  with relational scheme  $R(A)$  uniquely define them.
- More precisely, an attribute set  $X$  is a key for a relational scheme  $R(A)$  with F-dependencies  $F$  if it meets two conditions:
  1. Uniqueness:  $\{X \twoheadrightarrow A\} \in F^+$ .
  2. Minimality: for any subset  $Y$  of  $X$ ,  $Y \subset X$ , we have  $\{Y \twoheadrightarrow A\}$  does not belong to  $F^+$ .

# Normalization of relational schemas. Normal forms.

- Since it is possible several sets  $X$  that meet the uniqueness and minimality properties to exist, the key actually used in working with the relations is assumed to be the **primary key**, hereafter referred to simply as **key**.
- In the previous chapter, examples were discussed showing that there are relational schemes that are not "good". In the next section are presented methods to address the aforementioned disadvantages of relational schemas.

# NORMAL FORMS

- *The normalization of a relational schema is a process aimed at its transformation, in which certain constraints are imposed on the newly obtained relational schemas, eliminating some undesirable properties.*
- Consider the relations “EMPLOYEE”, “EMPLOYEE\_1”, “EMPLOYEE\_2” and “EMPLOYEE\_3”.
- By the information contained in them, we can assume them to be equivalent.
- Considered as mathematical objects, however, they are completely different.
- The first one is 2-membered and the others are respectively 3, 5 and 6-membered relations.



# NORMAL FORMS

EMPLOYEE	IDENTIFICATION NUMBER	NAME
	4705127680 5212066341 5008156874	Petar Ivanov Maria Hristova Ivan Pavlov

EMPLOYEE_1	Code	Date	Name
	7680 6341 6874	470512 521206 500815	Petar Ivanov Maria Hristova Ivan Pavlov



# NORMAL FORMS

EMPLOYEE_2	Code	Day	Month	Year	Name
	7680	12	5	47	Petar Ivanov
	6341	6	12	52	Maria Hristova
	6874	15	8	50	Ivan Pavlov

EMPLOYEE_3	Code	Day	Month	Year	First name	Last name
	7680	12	5	47	Петър	Иванов
	6341	6	12	52	Мария	Христова
	6874	15	8	50	Иван	Павлов

# NORMAL FORMS

- A relational scheme is in first normal form (1NF) if the domains of its constituent attributes are atomic (simple), or if the attributes are atomic and there are no repeated attributes.
- *Atomicity* is a concept that is determined in the specific case.
- For example, if the information that is processed in the EMPLOYEE relation is only about the employees' identification numbers and names, we should assume that it is in 1NF
- However, if in the course of working with this relation we need to use parts of the identification number, e. g. day, month or year, the identification number must be considered as a non-atomic (composite) attribute and therefore the relation EMPLOYEE will not be in 1NF.

# NORMAL FORMS

- In some applications, the attribute "Name" may be a composite attribute and its parts - first and last name are used.
- The relational scheme of the EMPLOYEE\_3 relation contains only atomic attributes and it is in INF. Its key is composed and includes the attributes (Code, Year, Month, Day).
- An attribute may also be non-atomic in the case where its elements are sets.
- Such is the case with the PERSONNEL relation. This relation is not in INF. Its normalization in INF is given by the PERSONNEL\_1 relation.

# NORMAL FORMS

PERSONNEL	Name	Education
	Petar Ivanov Maria Hristova Ivan Plavlov	{Secondary technical} {Mathematical, engineering} {Economic}

PERSONNEL_1	Name	Education
	Petar Ivanov Maria Hristova Maria Hristova Ivan Pavlov	{Secondary technical} {Mathematical} {Engineering} {Economic}

# NORMAL FORMS

- There may be various specific considerations for using normalized relations, i. e. , relations in INF.
- One of the most important is the need for accuracy in the representation of F-dependencies.
- For example, if the attribute “Star sign” is added to the relational schema EMPLOYEE\_1, then it is not possible to express the presence of the F-dependency:

(Day, Month)  $\rightarrow$  Star sign

# NORMAL FORMS

- This F-dependency, however, is represented in the relational schemes of the EMPLOYEE\_2 and EMPLOYEE\_3 relations

## Another example of normalization to 1NF:

EMPLOYEE (No, name, Child (firstname, age))

EMPLOYEE (No, name); CHILDREN( No, firstname, age)

No	NAME	CHILD	
		FIRSTNAME	AGE
500	DUPONT	ANDRE	10
501	DURAND	JEAN	11
501	DURAND	PIERRE	12
510	LEFEBVRE	PAUL	13
510	LEFEBVRE	JACQUES	14

EMPLOYEE	No	NAME
	500	DUPONT
	501	DURAND
	510	LEFEBVRE

CHILDREN	No	FirstName	age
	500	André	10
	501	Jean	11
	501	Pierre	12
	510	Paul	13
	510	Jacques	14

# NORMAL FORMS

- Let  $R(A)$  be a relational scheme with F-dependencies  $F$ . An attribute  $Y$  is said to be in *complete functional dependence* on an attribute  $X$ ,  $X, Y \in P(A)$  if  $\{X \twoheadrightarrow Y\} \in F^+$  and if for every subset  $X'$ ,  $\{X' \twoheadrightarrow Y\} \in F^+$  is satisfied.
- For example, the attribute “Star sign” is not in full F-dependence on the attributes (Day, Month, Year), but it is in full F-dependence on the attributes (Day, Month).



# NORMAL FORMS

- An attribute  $X$  of a relational scheme  $R(A)$  with an F-dependency  $F$  is called a *primary attribute* if it is part of the key. Otherwise it is called non-primary.
- Нека е дадена релацията ОБУЧЕНИЕ с релационна схема:  
EDUCATION (Faculty#, Name, City, Subject#, Mark)
- The key of this relation is composite (Faculty#, Subject#). The set of F-dependencies is:  
 $(\text{Faculty\#, Subject\#}) \rightarrow \text{Faculty\#, Name, City, Subject\#, Mark}$   
 $\text{Faculty\#} \rightarrow \text{Name, City}$

# NORMAL FORMS

EDUCATION	Faculty#	Name	City	Subjet #	Mark
	40385	Ivan Todorov	Pleven	M25	4.0
	40001	Teodosi Sabev	Kazanlak	M21	6.0
	41012	Rumyana Mineva	Kazanlak	C12	6.0
	40205	Hristo Petrov	Sofia	P09	5.5
	40385	Ivan Todorov	Pleven	M21	5.0
	40385	Ivan Todorov	Pleven	P09	6.0
	41012	Rumyana Mineva	Kazanlak	M25	5.5
	40205	Hristo Petrov	Sofia	M21	5.0
	41012	Rumyana Mineva	Kazanlak	M21	6.0

# NORMAL FORMS

- Obviously, the Name and City attributes are not in complete *F*-dependence of the key. What are the consequences of this?
  1. Until a student passes at least one exam, information pertaining to that student (Faculty#, Name, and City) cannot be entered because the value of the Subject# key attribute is undefined.
  2. The student's name and city are repeated as many times as the number of passed exams.
  3. If a student's only exam is cancelled, information about that student that is not related to that exam will be deleted as a consequence.

# NORMAL FORMS

- A relational scheme is in second normal form (2NF) on the set of  $F$ -dependencies  $F$  if it is in 1NF and every non-primary attribute is in full  $F$ -dependency on the key.
- The above disadvantages of such constructed relational scheme can be removed if instead we consider two relational schemes that are in second normal form (2NF).

STUDENT (Faculty#, Name, City);

STUD\_MARK (Faculty#, Subject#, Mark).

# NORMAL FORMS

- The F-dependencies in these relational schemes are:

Fac# ---> Name, City  
Fac#, Subject# ---> Mark

STUDENT	Fac#	Name	City
	40385	Ivan Todorov	Pleven
	40001	Teodosi Sabev	Kazanlak
	41012	Rumyana Mineva	Kazanlak
	40205	Hristo Petrov	Sofia

STUD_MARK	FAC#	SUBJECT#	MARK
	40385	M25	4.0
	40001	M21	6.0
	41012	C12	6.0
	40205	P09	5.5
	40385	M21	5.0
	40385	P09	6.0
	41012	M25	5.5
	40205	M21	5.0
	41012	M21	6.0

# NORMAL FORMS

- In the resulting relations each non-primary attribute is in full  $F$ -dependence on the key.
- Let  $R(A)$  be a relational scheme with a corresponding set of  $F$ -dependencies  $F$ .
- It is assumed that  $X$  and  $Z$  are subsets of  $A$ . An attribute  $Z$  is said to be transitively dependent on  $X$  if there exists such set of attributes  $Y$ ,  $Y \subset A$ ,  $Z$  does not belong to  $XY$ , for which the following properties applies:

$$\begin{aligned}\{X \multimap Y\} &\in F^+ \\ \{Y \multimap X\} &\in F^+ \\ \{Y \multimap Z\} &\in F^+ .\end{aligned}$$

# NORMAL FORMS

- **Example of transitive dependence**

Auhtors			
Author_ID	Author	Book	Author_Nationality
Auth_001	Orson Scott Card	Ender's Game	USA
Auth_001	Orson Scott Card	Ender's Game	USA
Auth_002	Margaret Atwood	The Handmaid's Tale	Canada

In the above example **AUTHORS: Book → Author**: Here, the Book attribute defines the Author attribute. If you know the name of the book, you can learn the name of the author. However, an author does not define a Book, because an author can write several books. For example, just because we know the name of the author, Orson Scott Card, we still don't know the name of the book.

- **author → Author\_Nationality** : Similarly, the author attribute defines Author\_Nationality, but not vice versa; just because we know the nationality doesn't mean we can identify the author.

But this table introduces transitive dependence:

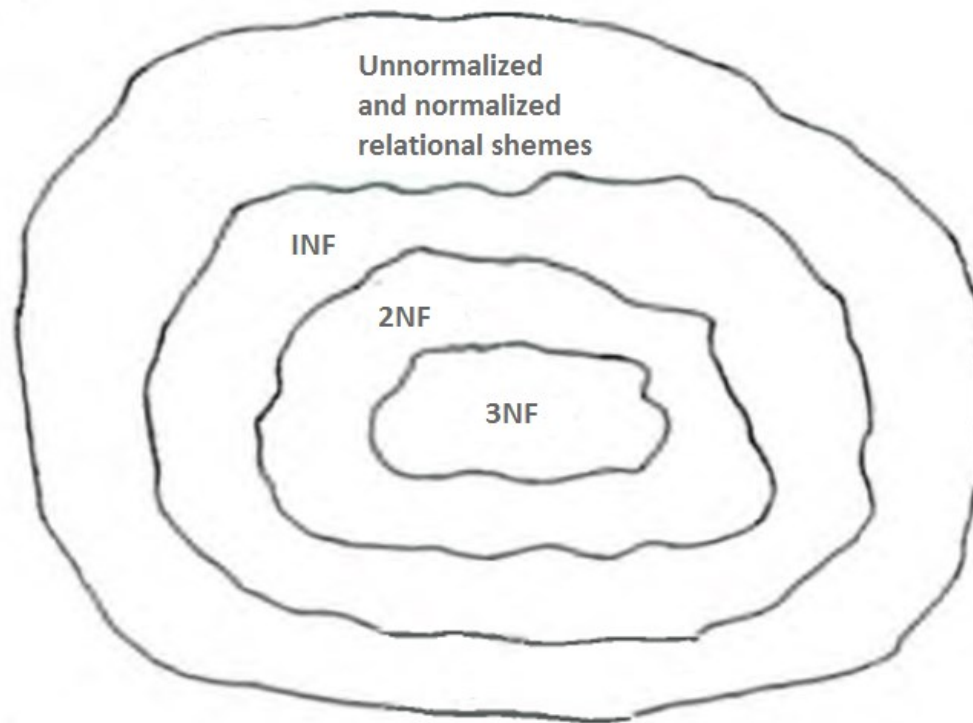
**Book → Author\_Nationality**: If we know the name of the book, we can define the nationality using the "Author" column.



# NORMAL FORMS

- A relational scheme  $R(A)$  is in **third-normal form (3NF)** on the set of F-dependencies  $F$  if it is in INF and no non-primary attribute is transitively dependent on the key.
- The definition of 2NF requires the presence of INF, i. e. any relational scheme that is in 2NF is also in INF.
- The definition of the 3NF does not require the relational scheme to be in 2NF, but it can be shown that any relational scheme that is in 3NF is also in 2NF.

# NORMAL FORMS



**Input of the relation sets in 3NF, 2NF and 1NF**

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# NORMAL FORMS

- Suppose that the relational scheme  $R(A)$  is in 3NF but not in 2NF. Let  $Z \in P(A)$  and  $Z$  is not in complete F-dependence on the key  $K$  of  $R(A)$ .
- This means that a subset  $K'$  of  $K$  exists for which the F-dependence  $K' \twoheadrightarrow Z$  applies.
- But since  $K$  is a key it follows that:

$$\{K \twoheadrightarrow K'\} \in F$$

$$\{K' \twoheadrightarrow K\} \notin F^*$$

# NORMAL FORMS

- Finally, a transitive  $F$  –dependence is achieved, i. e.  $R(A)$  is not in 3NF, which contradicts with the assumption made above.
- The opposite statement is not true. For example, the STUDENT\_2 relation is in 2NF but not in 3NF.

STUDENT_2	Fac_num	Name	City	Region
	40385	Ivan Todorov	Pleven	Lovech
	40001	Teodosi Sabev	Kazanlak	Haskovo
	41012	Rumyana Mineva	Kazanlak	Haskovo
	40205	Hristo Petrov	Sofia	Sofia

# NORMAL FORMS

- The reason for this are the F-dependencies:

Fac\_numb - - - - > Name, City, Region;

City - - - - > Region.

from which follows the transitive F-dependence



# NORMAL FORMS

- What are the effects of having this transitive dependence?
  1. It is not possible to register the fact that the town of Bansko is from the Blagoevgrad region if not a single student is from the town of Pleven.
  2. If for some reason the students from Kazanlak move to another faculty, then the information that the town of Kazanlak is from Stara Zagora region will be lost.
  3. If a new territorial zoning needs to be made, where the town of Kazanlak will be the regional center, then in the relation STUDENT\_2 will be needed as many changes as all the students from Kazanlak. And this leads to a waste of time and creates conditions for introducing inconsistencies in the data, i. e. there are conditions for violating the integrity of the data in the DB.

# NORMAL FORMS

- The above mentioned disadvantages of the STUDENT\_2 relational scheme are due to **transitive dependency** and are avoided by removing it.
- This is done in the schemes of the STUDENT\_3 and ZONNING relations, which are derived from STUDENT\_2. These two relational schemes are in the 3NF.

СТУДЕНТ_3	Факт-ном	Име	Град
	40385	Иван Тодоров	Плевен
	40001	Теодоси Събев	Казанлък
	41012	Румяна Минева	Казанлък
	40205	Христо Петров	София

РАЙОНИРАНЕ	Град	Област
	Плевен	Ловеч
	Казанлък	Хасково
	София	София



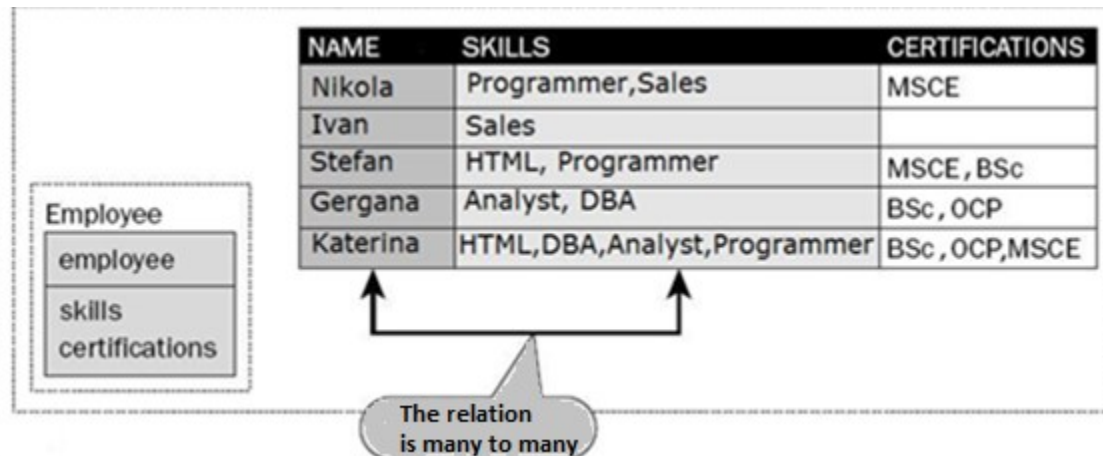
# NORMAL FORMS

## Fourth normal form (4NF)

- A relation is in 4NF only if it does not contain multi-valued dependencies.
- A multi-valued dependency exists when there are three attributes (A, B and C) in a relation and for each value of A there is a well-defined set of values B and a well-defined set of values C. The set of values B is independent of C and vice versa.

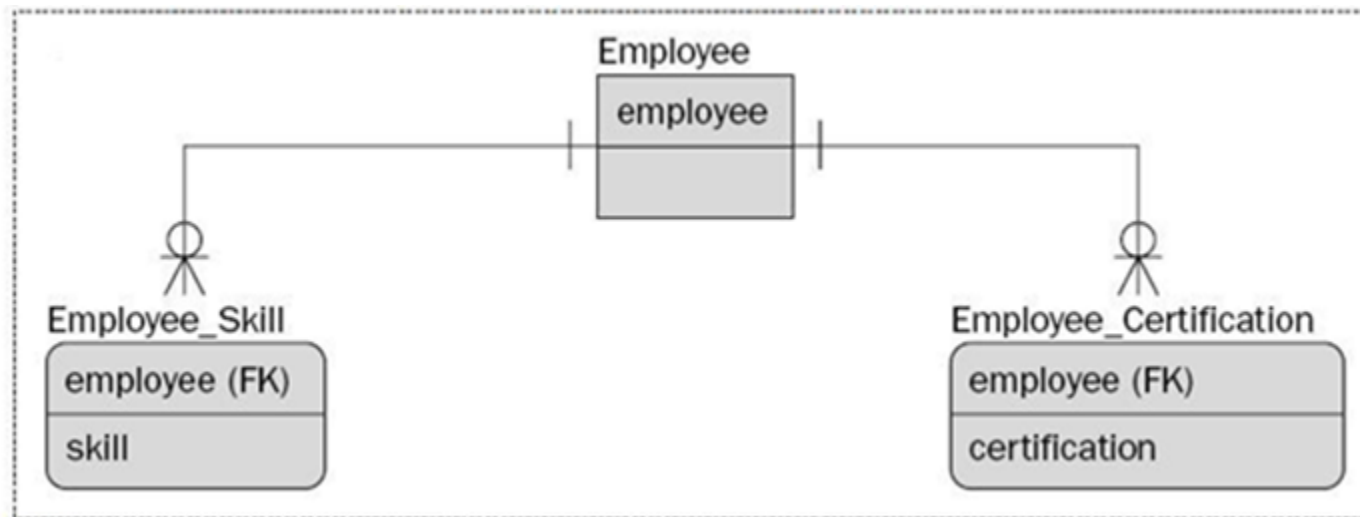
# NORMAL FORMS

- A multivalued field is a field that contains a collection of an array of values of some type separated by commas. The figure below shows the Employee table with the different skills that employees have and their certificates. It should be noted that the skills and certificates are not only independent of each other but are added in a list separated by commas.



# NORMAL FORMS

- To apply the 4th normal form we remove the multi-valued dependencies by splitting the Employee table into three tables Employee\_Skill, Employee and Employee\_Certification:



- Essentially, the 4th normal form transfer the collection of multi-valued elements into different tables and records and thus makes each record easier for direct access.

# CONCLUSION

- The normalization of relational schemes is directly related to the semantics of the data in the DB, not to the values and contents of specific relations.
- It is not possible to establish functional dependencies by the content of a particular relation, and this means that it is not possible to establish its normal form.
- The DBMS is the one that must maintain data integrity by taking into account its specified functional dependencies.
- However, if a relational scheme is in a 3NF, it is enough for the DBMS to know only the key of the corresponding relation to ensure the uniqueness of the elements in each relation.

# CONCLUSION

- If the relational scheme is not in the 3NF, then storing and maintaining the F-dependencies by the DBMS is needed. This was shown with concrete examples.
- The above considerations justify the need to analyse relational schemes and in particular to send them into the **3NF**.

*In the next section, two approaches for the normalization of relational schemes are discussed.*

# Questions and exercises:

1. What is Normalization?
2. What are the various forms of Normalization?
3. What is First Normal Form?



# Thank you for your attention!



# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 2. Relational approach in databases

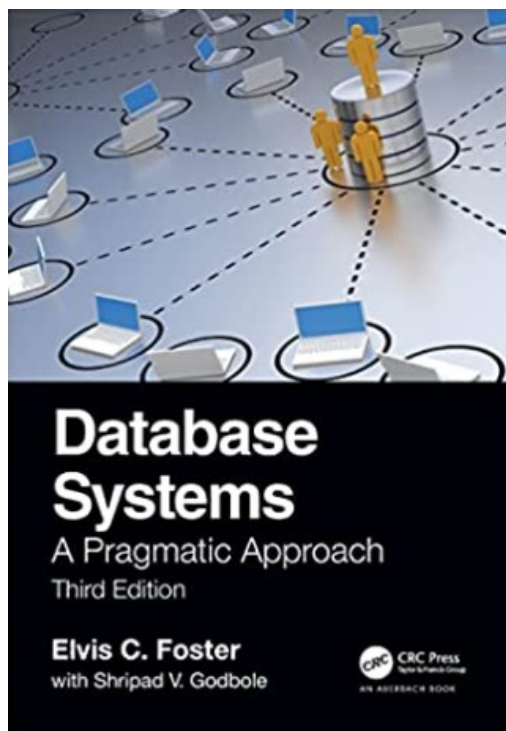
### ❑ Topic 4. Development of a sample database. Normalization.

- ❑ Lesson 2. Decomposition and synthesis of relational schemas. Normalization algorithm by decomposition. Algorithm for synthesis of relational schemas.

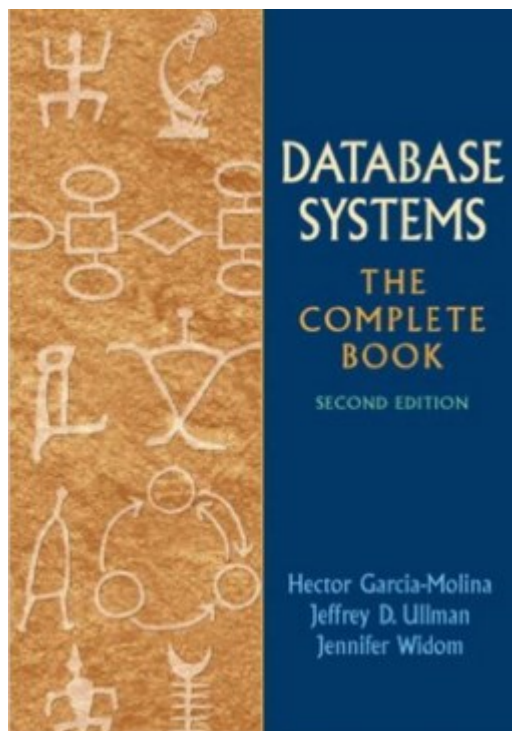


ERASMUS+

# DECOMPOSITION AND SYNTHESIS OF RELATIONAL SCHEMAS. NORMALIZATION ALGORITHM BY DECOMPOSITION. ALGORITHM FOR SYNTHESIS OF RELATIONAL SCHEMAS.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# Normalization algorithm by decomposition

- **Two main methods are used for the normalization of relations - decomposition and synthesis.**

## Normalization algorithm by decomposition

- Let  $R(A)$  be a relational schema. To perform a decomposition on  $R$  means to define a set of relational schemas of the form:

$$R' = \{R_1(A_1), R_2(A_2), \dots, R_k(A_k) \mid A = \bigcup_{i=1}^k A_i\}$$

- In other words, the new relational schemas have as attributes some of the attributes of  $R$ , and their union gives the set of attributes  $A$ , and the intersection of the attributes of any two relational schemas can be either the empty set or the non-empty set.

# Normalization algorithm by decomposition

- The point of the idea to perform relational schema decomposition is the need to remove unwanted anomalies, such as *redundancy anomalies, update, insertion and deletion anomalies*.
- Let  $R(A_1, A_2, A_3, A_4)$  be a given relational schema, and let  $R'$  be its decomposition  $R' = \{R_1(A_1, A_2), R_2(A_1, A_3, A_4)\}$ . It is also assumed that:
  1.  $r$  is a relation with relational schema  $R(A_1, A_2, A_3, A_4)$
  2.  $r_1 = \pi_{A_1 A_2}(r)$ , t.e.  $r_1: R_1(A_1, A_2)$
  3.  $r_2 = \pi_{A_1 A_3 A_4}(r)$ , t.e.  $r_2: R_2(A_1, A_3, A_4)$

# Normalization algorithm by decomposition

- It is natural to ask whether, under the assumptions so formulated, the relation  $r$  can be said to be recoverable under its projections  $r_1$  and  $r_2$ .
- Let  $s = r_1 \bowtie_{A_1} r_2$ , i. e.  $s$  is the natural union of  $r_1$  and  $r_2$  with respect to the attribute  $A_1$ . If  $r=s$  for any  $r$  with relational schema  $R(A)$  and corresponding set of functional dependencies  $F$ , then the decomposition of  $R(A)$  is said to have the lossless conjunction property on  $F$ .

# Normalization algorithm by decomposition

- Let  $R(A)$  be a relational schema with set of functional dependencies  $F$  and  $R' = \{R_1(A_1), R_2(A_2)\}$  be a decomposition of it.
- If  $r$  is a relational schema  $R(A)$  and  $r_i$  is a relational schema  $R_i(A_i)$ , where:  $r_i = \pi_{A_i}(r)$ ,  $i = 1, 2$ .

than

$$1. r \subseteq r_1 \bowtie r_2$$

$$2. r = r_1 \bowtie r_2, \text{ only if}$$

$$\{A_1 \cap A_2 \dashrightarrow A_1 \setminus A_2\} \in F^+ \quad \text{or}$$

$$\{A_1 \cap A_2 \dashrightarrow A_2 \setminus A_1\} \in F^+$$



# Normalization algorithm by decomposition

- We will illustrate this with some examples.
- The relational schema STUDENT (Fac\_\_numb, Name, Address) is considered, where the attribute Address denotes the address where the student lives during the study.
- We note that several students may live at the same address and that a student may change addresses several times during his/her studies.
- This gives a reason for specifying a single functional dependency in the STUDENT relational schema:

$$F = \{\text{Fac\_numb} \twoheadrightarrow \text{Name}\}$$



# Normalization algorithm by decomposition

- The following decomposition is considered:

$$R' = \{R_1(\text{Fac\_numb}, \text{Name}), R_2(\text{Fac\_numb}, \text{Address})\}$$

- Let STUD be a relation with relational schema STUDENT, and STUD1 and STUD2 be its projections.

$$\begin{aligned} \text{STUD1} &= \pi_{\text{Fac\_numb}, \text{Name}} (\text{STUD}) \\ \text{STUD2} &= \pi_{\text{Fac\_numb}, \text{Address}} (\text{STUD}) \end{aligned}$$

STUD	Fac_num	Name	Address
	40251	Ivan Koychev	Sofia, Stud.grad, bl.25, A, room 128
	40326	Petar Dimov	Sofia, Stud. grad, bl.25, A, room 128
	40138	Ivan Koychev	Sofia, Vitosha bul, № 18, ap. 42

# Normalization algorithm by decomposition

STUD1	Fac_num	Name
	40251 40326 40138	Ivan Kouchev Petar Dimov Ivan Koychev

STUD2	Fac_num	Address
	40251 40326 40138	Sofia, Stud. grad, bl. 25, A, room 128 Sofia, Stud. grad, bl. 25, A, room 128 Sofia, Vitosha bul, № 18, ap. 42

# Normalization algorithm by decomposition

- Let  $S$  be the natural union of STUD1 and STUD2, i. e.

$$S = \text{STUD1} \bowtie \text{STUD2}$$

**Fac.number**

- It can be seen that  $S = \text{STUDENT}$ . This is not accidental, since there are conditions for lossless join decomposition on  $F$ .

**Really:**

$$\begin{aligned} \{ \text{Fac\_numb}, \text{Name} \} \cap \{ \text{Fac\_numb}, \text{Address} \} &\simeq \text{Fac\_numb} \\ \{ \text{Fac\_numb}, \text{Name} \} \setminus \{ \text{Fac\_numb}, \text{Address} \} &\simeq \text{Name} \\ \{ \text{Fac\_numb} \twoheadrightarrow \text{Name} \} &\in F^+ \text{ condition} \end{aligned}$$

# Normalization algorithm by decomposition

- Another decomposition of the STUDENT relational schema is as follows:

$$R' = \{R_1(\text{Fac\_numb}, \text{Name}), R_2(\text{Name}, \text{Address})\}$$

- Let again STUD be a relation with relational schema STUDENT, and the relations STUD1 and STUD3 are defined as projections of STUD as follows:

$$\begin{aligned}\text{STUD1} &= \pi_{\text{Fac\_numb}, \text{Name}}(\text{STUD}) \\ \text{STUD3} &\approx \pi_{\text{Name}, \text{Address}}(\text{STUD})\end{aligned}$$

# Normalization algorithm by decomposition

STUD3	Name	Address
	Ivan koychev	Sofia, Stud. grad, bl. 25, A, room 128
	Petar Dimov	Sofia, Stud. grad, bl. 25, A, room 128
	Ivan Koychev	Sofia, Vitosha bul., № 18, ap. 42

- The natural union of STUD1 and STUD3 is the STUD № relation:

$$STUD_X = STUD1 \bowtie_{Name} STUD3$$

STUD <sub>X</sub>	Fac_num	Name	Address
	40251	Ivan Koychev	Sofia, Stud.grad, bl. 25, A, room 128
	40251	Ivan Koychev	Sofia, Vitosha bul., № 18, ap. 42
	40326	Petar Dimov	Sofia, Stud.grad, bl. 25, A, room 128
	40138	Ivan Koychev	Sofia, Stud.grad, bl. 25, A, room 128
	40138	Ivan Koychev	Sofia, Vitosha bul., № 18, ap. 42

# Normalization algorithm by decomposition

- It can be seen that STUD<sub>1</sub> and STUD are not identical. And this fact is not accidental. It is enough to notice that:

$$(Fac\_numb, Name) \cap (Name, Address) = Name$$

$$(Fac\_numb, Name) / (Name, Address) = Fac\_numb$$

$$(Name, Address) / (Fac\_numb, Name) = Address$$

$$\{ Name \dashrightarrow Fac\_numb \} \notin F^+$$

$$\{ Name \dashrightarrow Address \} \notin F^+$$

- Decomposition basis is in transforming relational schemas that are not in the 3NF with respect to a set of functional dependencies  $F$  into relational schemas that are in 3NF.*

# Normalization algorithm by decomposition

- This transformation is done in two steps. In the first step, the relational schemas are brought to 2NF, and in the second step to 3NF. The idea of bringing relational schemas into 2NF will be discussed first.
- Let  $R(A)$  be a given relational schema which is in 1NF and not in 2NF. This means that:
  - a) the key of this relational schema is composite, i. e. it consists of at least two attributes, which we denote by  $K_1$  and  $K_2$ ,  $K_1 \in A$  and  $K_2 \in A$ .
  - b) There are two subsets of  $A$  which we denote by  $X$  and  $Y$ ,  $X \subset A$ ,  $Y \subset A$ , as  $(K_1, K_2) \twoheadrightarrow Y$  and  $K_1 \twoheadrightarrow X$ , i. e. the attribute  $X$  is not in full functional dependence on the key.



# Normalization algorithm by decomposition

- From the above explanations follows that it is possible to replace the relational schema  $R(K_1, K_2, X, Y)$ , which is not in 2NF, by two relational schemas:

$$R_1(K_1, K_2, Y) \text{ and } R_2(K_1, X)$$

- If  $R_1$  and  $R_2$  are not in 2NF, the described procedure must be applied again. This continues until the original relational schema is transformed into a set of relational schemas, all of which are in 2NF. We denote a normalization procedure in 2NF by:

$$\text{DECOMP-2NF}(A, R_1, R_2, F)$$

# Normalization algorithm by decomposition

## WHERE

- $A$  - the set of attributes in the original relational schema;
- $R_1, R_2$  - relational schemas on which the original relational schema is decomposed;
- $F$  - the set of functional dependencies in  $A$ .
- To bring relational schemas into the 3NF, the following idea can be applied.
- Let  $R(A)$  be a given relational schema, with F-dependencies  $F$ , which is in 2NF but not in 3NF.
- This means that there are transitive functional dependencies in  $R(A)$ , i. e. , the set of attributes  $A$  can consists of  $K, X_1, X_2$  and  $X_3$ , where  $K$  is a key and  $X_1, X_2, X_3$  are non-key attributes for which  $X_1 \twoheadrightarrow X_2$  and  $\{X \twoheadrightarrow K\} \in F^+$ . It is not excluded that the  $X_3$  attribute does not exist.

# Normalization algorithm by decomposition

- Under these assumptions, the original relational schema  $R(K, X_1, X_2, X_3)$  can be decomposed into two relational schemas  $R_1(K, X_1, X_3)$  and  $R_2(X_1, X_2)$ . Let denote the normalization procedure from 2NF to 3NF by:  $\text{DECOMP-3NF}(A, R_1, R_2, F)$
- Consider the full *normalization algorithm by decomposition*. It denotes by  $\text{NORM\_2}(RA, F)$  and  $\text{NORM\_3}(RA, F)$  two Boolean-type functions that check whether the relational schema  $RA$  is in second and third normal form, respectively.

# Normalization algorithm by decomposition

```
const N = ...;
type A = (A1,A2,...,Ak);
    Attr = set of A;
    F-Dep = record
        X,Y:Attr
    end;
    FD = array[1..N] of F-Dep;

procedure DECOMP(RA:A;F:FD);
    var R1,R2 : A;
    begin
        if not NORM-2(RA;F)
        then begin
            DECOMP-2NF(RA,R1,R2,F);
            DECOMP(R1,F);
            DECOMP(R2,F)
        end
        else begin
            if not NORM-3(RA,F)
            then begin
                DECOMP-3NF(RA,R1,R2,F);
                DECOMP(R1,F);
                DECOMP(R2,F)
            end
        end
    end;
end;
```

# Normalization algorithm by decomposition

- The algorithm for normalization of relational schemes by decomposition has some **disadvantages**.
- The complexity of this algorithm is too big. This is because it may be that the number of keys in a relational schema grows exponentially with the number of attributes and the number of functional dependencies.
- No simpler is the task of checking the **primality** of an attribute.
- The decomposition can sometimes result in a much larger set of relational schemas in the 3NF than necessary.
- These considerations give reason for consideration of **one more** algorithm for the normalization of relational schemas.

# Algorithm for relational schemas synthesis

- Briefly, the idea of the relational schema synthesis algorithm is the following:
  1. The set of functional dependencies  $F$  is considered, and the individual dependencies are grouped by equality on their left sides.
  2. For each group obtained from point1, a separate relational schema with a set of attributes is defined obtained from the union of the attributes taking place in the functional dependencies of each individual group.

**Example.** Consider the set of attributes  $\{A, B, C\}$  and the set of functional dependencies  $F = \{A \twoheadrightarrow B, A \twoheadrightarrow C, B \twoheadrightarrow C\}$ .

# Algorithm for relational schemas synthesis

- From point 1 of the algorithm, two sets of F-dependencies are obtained:

$$\begin{aligned} \{ A \multimap B, A \multimap C \} &\longrightarrow R_1(A, B, C) \\ \{ B \multimap C \} &\longrightarrow R_2(B, C) \end{aligned}$$

- It is immediately apparent that it is not in 3NF, since  $A \multimap B$ ,  $B \multimap C$  follows  $A \multimap C$ , i. e. there is a transitive dependence.
- The incorrect result that is received is due to an excess of F-dependencies. To remove this incorrectness, some refinement of the original algorithm idea is required.



# Algorithm for relational schemas synthesis

- Let  $f = \{X \twoheadrightarrow Y\}$  be a functional dependence of  $F$ . Attribute  $X$  is called a non-redundant attribute with respect to  $f$  if  $Y$  is in complete functional dependence on  $X$ , i. e. if for any  $X'$ ,  $X' \in X$ , it follows that:

$$((X - \{X'\}) \twoheadrightarrow Y) \notin F^+$$

- Let's remind that a set of functional dependencies  $F$  is **redundancy-free** if for any  $f$ ,  $f \in F$  it follows that:

$$(F - \{f\})^+ \neq F^+$$

- These two notions are substantially involved in specifying the idea of synthesis of normalized relational schemas, so they are described as procedures.

# Algorithm for relational schemas synthesis

- The REDUNDANT\_ATR procedure removes all redundant attributes from the composite attribute  $B_l$  with respect to the functional dependency  $B1 \twoheadrightarrow B2$ . This procedure uses the IS\_MEMBER function, which determines the membership of a functional dependency to a set of functional dependencies.

```
const  N = ...;  
       K = ...;  
type   A = (A1,A2,...,Ak);  
       Attr = set of A;  
       F-Dep = record  
           X,Y:Attr  
       end;  
       FD = array[1..N] of F-Dep;
```

# Algorithm for relational schemas synthesis

```
procedure REDUNDANT-ATR(var B1:Attr;B2:Attr;F:FD);
  var I : integer;
      W : Attr;
      Z : A;
      f : F-Dep;
  begin
    Z:=A1;
    for I:=1 to K do
      begin
        if Z in B1
          then begin
            W:=B1-[Z];
            f.X:=W;
            f.Y:=B2;
            if IS-MEMBER(f,F)
              then B1:=B1-Z
            end;
            Z:=succ(Z)
          end
        end
      end;
  end;
```

# Algorithm for relational schemas synthesis

- Applying the REDUNDANT\_ATR procedure to the left side of each of the functional dependencies of  $F$  a set of functional dependencies  $G$  is obtained, whose left sides are attributes without redundancy. We denote this new procedure by REDUNDANT ALL ATR( $F, G$ ).
- By analogy to the procedure REDUNDANT\_ATR, a procedure REDUNDANT\_FDEP( $G, H$ ) can be created that removes redundant functional dependencies from  $G$  and results in a new set of functional dependencies  $H$  that is equivalent to  $G$ , i. e.  $G^+ = H^+$ .

Now the *algorithm for synthesizing relational schemas* in 3NF itself can be described.

# Algorithm for relational schemas synthesis

1. The procedure REDUNDANT\_ALL\_ATTR( $F, G$ ) is applied.
2. The procedure REDUNDANT\_FDEP( $G, H$ ) is applied.
3. The set of functional dependencies  $H$  is divided into groups, each containing functional dependencies with the same left side.
4. From each group of functional dependencies, a relational schema is defined that contains the union of the attributes appearing on the left and right sides of the functional dependencies. The keys of the relational schemas are chosen to be the attributes that are on the left side of the functional dependencies of each group.

# Questions and exercises:

1. What is decomposition of a relation schema ?
2. Which is the properties of a relational decomposition ?
3. What is synthesis of a relation schema ?



# Thank you for your attention!



# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 2. Relational approach in databases

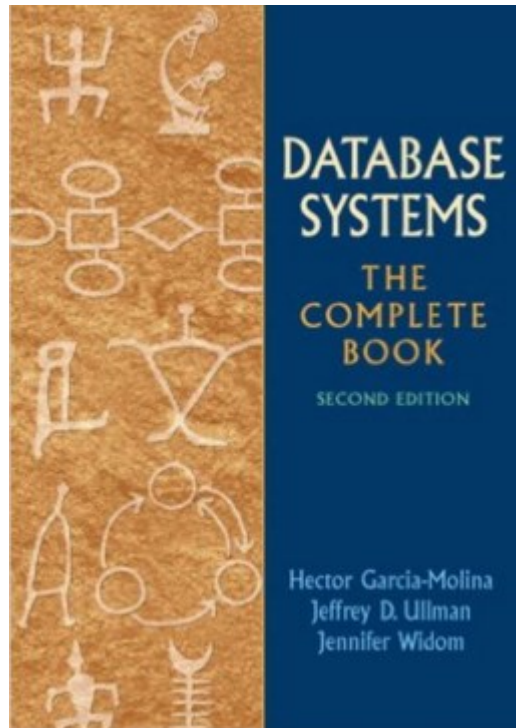
### ❑ Topic 5. Object-oriented database systems.

- ❑ Lesson 1. The essence of objects. Basic concepts in the object-oriented approach.

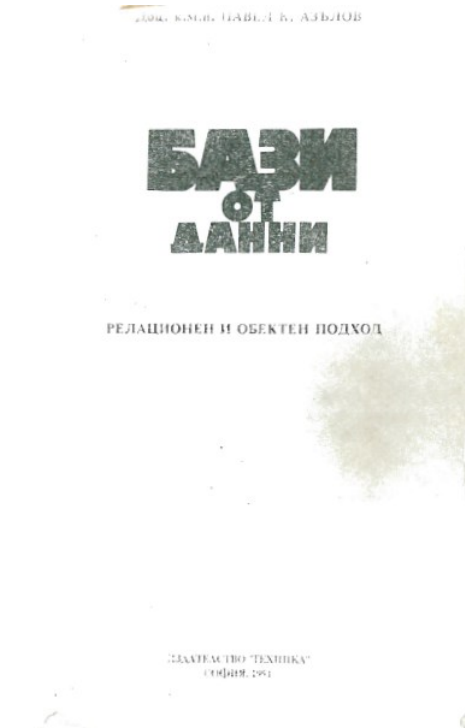


ERASMUS+

# THE ESSENCE OF OBJECTS. BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH.



Ullman, J., Widom, J (2009) DATABASE  
SYSTEMS The Complete Book (2rd ed),  
Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# OBJECT NATURES

- In general, the word object can refer to any object: a book, a building, a car, an enterprise, a city, a computer, etc. Objects may or may not be material. For example, the liberation wars fought in Europe in the 19th and 20th centuries can be seen as events that are historical objects.
- In other words, the object everything that can be directly or indirectly referred to. Similar generalizations are made in various fields of science.
- Here are some examples from informatics:
  1. The point, segment, vector, arc, circle, broken line, etc. are often summarized by the name **graphical object**.

# OBJECT NATURES

2. Integer, real quantities are **simple objects**. Sign and Boolean quantities are **simple objects**.
3. Arrays, character strings, records, sets, files, etc. are **composite objects**.
4. The algorithmic constructs sequence, iteration (loop), binary and multivariate choice are **control objects** in the programs.
5. A subprogram can also be viewed as an object - a **program object** that generalizes the objects described above and that includes a description of only *data objects* or only *operations objects* of some objects or contains objects of both types at the same time.

# OBJECT NATURES

6. The text fragment is a **text object**, the graphic image is a **graphic object**, the tables and the form are also objects and all of them separately and in a certain combination can make up the more complex **information object** - the document

- This list of examples could be extended. However, it is good to look into the meaning that is put into each of these generalizations and the benefit that can be derived from it.
- It is therefore natural to start by introducing the **concept of object**.

# OBJECT NATURES

- **The object** consists of matter represented by some data types, which is "spiritualized" by embedding in it certain behavioral rules.
- Like living organisms, objects exist in a particular environment, which in this case is provided by *relevant software, computer and telecommunication systems*.
- Objects can be *active* (assumed alive) or *inactive* (assumed not alive). Active objects participate in a variety of events, such as the "birth" of a new object or the "move" of an object from one location to another.

# OBJECT NATURES

- They "communicate" by exchanging information by sending messages both to each other and to the rest of the world - to users.
- All of this active life is performed by objects under the guidance of a "**superobject**", which is further referred to by the abbreviation **OM (Object Manager)**.
- **The OM** is a software subsystem that, according to the behavior embedded in the objects, manages and controls their actions.
- Such an intermediate level between user and object frees the user from knowing the specific details pertaining to individual objects.



# OBJECT NATURES

- In "anatomical" terms, each object is represented by an identification part, a body and a behavior. The identification part, also called "head", identifies the object among other objects.
- The body of each object is a set of named attributes. Here the variety is too big.
- The objects may be simple or may themselves include other objects, thus representing a **objects hierarchy**. Each object is characterized by a behavior represented by a set of rules. The rules resemble subprograms.

# OBJECT NATURES

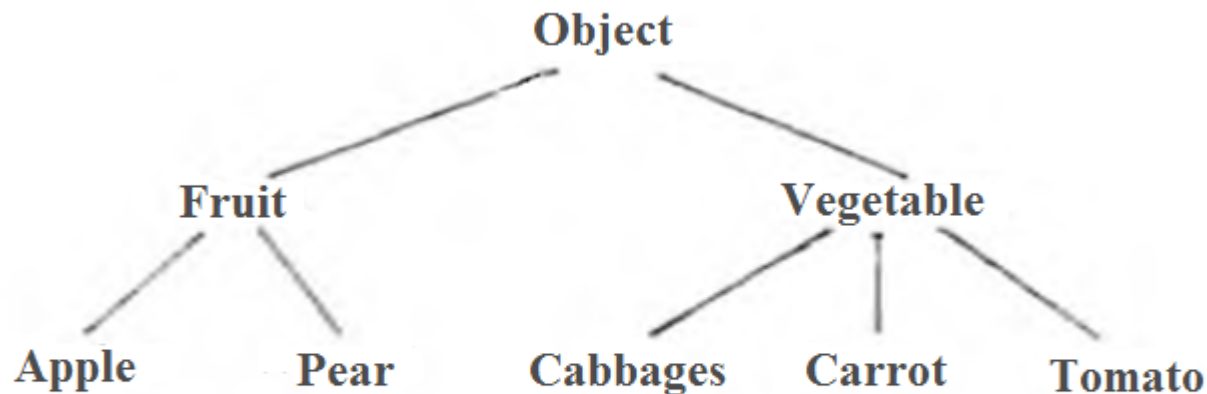
- However, there is no basic program between them to activate them. The rules use object-local variables as well as conditions under which these rules can be activated.
- Solving a task using the object-oriented approach requires the creation of different objects.
- Some of them will have similar properties relating to the structure of the data and the corresponding operations applied to it. This is a prerequisite for grouping such objects into groups called **classes**.
- Thus, each object can be viewed as a separate sample of some class.

# OBJECT NATURES

- The introduction of the class concept creates conditions for a single definition of the properties of objects of the same class.
- In nature, each object is usually individual, but it also has common features with similar objects.
- *For example*, each apple in an *apple* basket differs somehow from the others, but the recipes (methods) for making different types of nectars are the same for all the apples in the basket or outside it
- In this sense, the set of all apples can be seen as a separate class of objects different from, for example, the class of *pears*.

# OBJECT NATURES

- The process of grouping objects into classes is a type of *abstraction* and in this case is known as **generalization**.



**Fig. Example scheme of generalization of several object types**

- The highest in the hierarchy is the **Object** class, which is the generalizing class for all other classes.

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- Main concept in the object-oriented approach is the notion of **object**. This concept is difficult to be defined because it varies too much from one object-oriented system to another. Therefore, most often each author specifies it and gives it a specific meaning.
- The object is information and a description of its processing.
- It can be said that an object is an "encapsulated" unit of data and operations (methods) for processing it.
- The application of an operation does not require knowledge of how to implement it.

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- Each object can be considered as a value of some type. Let  $T$  be a set of some data types and let  $T1$  and  $T2$  be two types of  $T$ .
- It is said that  $T1$  is a **subtype** of  $T2$  or also that  $T2$  is a **supertype** of  $T1$  if every element (value) of type  $T1$  can be considered as an element of  $T2$ .
- **The subtype  $T1$**  inherits the properties of its **supertype  $T2$** . This means that the data structures and operations defined in  $T2$  are data structures and operations for the type  $T1$ .
- One type can be defined as a subtype of one only other type.

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- In such a case, we are talking about *simple heredity*.
- It is also possible for one type to be a subtype of several other types, and then we are talking about *multiple inheritance*.
- The heritability in the set of types  $T$  is a binary relation, which is often denoted by IsA. The properties of asymmetry, non-reflexivity and transitivity are valid for it.
- The IsA relation defines in  $T$  a hierarchy, often called a **type-hierarchy**.
- The term **class** is often used with the meaning of type in object-oriented DBs. **Class** has a different meaning.



# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- Each **class** is a set of objects of the same type which is time-varying. The object classes model sets of entities of different subject areas of the real world.
- In the set of classes  $C$ , a binary relational inheritance can be defined (denoted by `SubsetOf`) for which the asymmetry, nonreflexivity, and transitivity properties are valid. If a class  $C1$  is a subclass of  $C2$ , then:
  1.  $C1$  is a subset of  $C2$ .
  2. The  $C1$  element type is a subtype of the  $C2$  elements.

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- The SubsetOf relation defines a hierarchy in C called a **hierarchy of classes** or just a **class-hierarchy**.
- Specific to object-oriented DBs is the concept of **persistence**. **Persistence** is a property of any object that determines its duration of existence.
- In programming languages, objects are usually created and exist within the implementation of the corresponding program.
- The requirement for a system to maintain the **persistence** of the objects being created while the system itself exists is an important prerequisite for extending it with **object base management** capabilities.

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- Objects cannot activate themselves. They are activated "externally" by messages. The messages are similar to function and procedure calls in procedure-oriented systems. They are expressions pointing:
  - the recipient object of the message;
  - the content of the message.
- The content of the message is an instruction to apply an operation on the recipient object given a list of arguments (objects).

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- It is natural to apply the same operation to objects of different classes. For example, it is very convenient to use the "+" operation for both integers and real numbers, or to use the read and write procedures for quantities of different types. This idea is widespread in object-oriented systems and is known as polymorphism.
- **Polymorphism** allows the same messages to be sent to different objects, each "responds" in a manner consistent with its nature.
- Each object is distinguishable from the others in an object environment. This is done by an *identifier* that uniquely identifies the individual object.

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- The identification of an object does not depend on its current content, nor on its location, nor on the way it is accessed.
- Maintaining object identification is a characteristic feature of any object system. In general, the main characteristics of object-oriented systems can be summarized as follows:
  1. Solving a task in the environment of an object-oriented system requires:
    - a) determining the object classes required for the specific task;

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- b) defining the required messages to the different types of objects;
  - c) composing a sequence of messages that solves the task.
2. Operations consist of sending messages. The result of sending a message to an object is also an object.
  3. Options are available for defining object classes that inherit the properties of their superclass.
  4. Object-oriented systems are extensible. System and user classes have equivalent status.

# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

- There are systems that satisfy only some of the above requirements that are characteristic of object-oriented systems.
- If a system allows defining and operating with objects, it is accepted to call it an *object-based system*.
- If an object-based system allows the grouping of objects into separate classes, then it is a *class-based system*.
- It is obvious that **object-oriented systems** are a subset of **class-based systems**, which in particular are a subset of **object-based systems**.



# BASIC CONCEPTS IN THE OBJECT-ORIENTED APPROACH

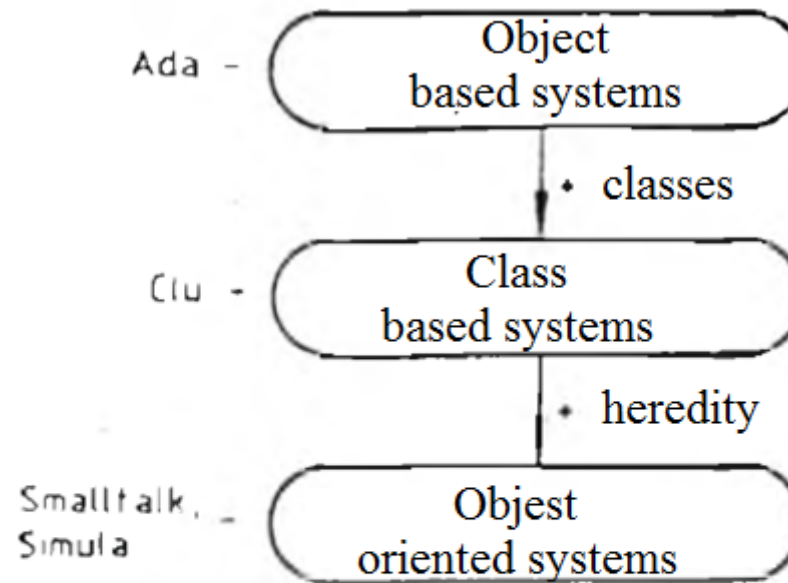


Fig. From object-based to object-oriented systems (P.Wegner, OOPSLA, 1983-1984)

# CONCLUSION

- The links between classes in databases, that can be expressed, are static and very limited. They mostly refer to the contents of individual attributes, and in relational systems they are implemented by *compound* of two or more tables.
- Links between individual instances, i. e. table rows, cannot be expressed. Links are made only at the entire table level.
- Unlike the Database approach, the essence of the object-oriented approach is the idea that an individual object existing in an environment can interact with other objects by establishing links with them.

# CONCLUSION

- A brief review of the basic capabilities of computer systems for storage and searching shows that their development is directly related to the need to extend the range of these systems, thus diversifying the subject areas in which they can be applied.
- This leads to the need to create computer models of complex real-world objects, requiring corresponding complex data structures to represent them.
- One possible approach to solving these problems is the object-oriented approach.

# CONCLUSION

- It become obvious that there are significant and varied points of incompatibility between the Database Approach and the Object-Oriented Approach. Nevertheless, it turns out that they can and are already being applied together.
- In object-oriented databases, it is proposed to extend the DBMS with object-oriented concepts.
- Such can be introduction of abstract data types, handling complex objects, and the use of inheritance generated by the class-hierarchy of object classes.

# CONCLUSION

- A significant advantage of the approach is that it remains close to the "Databases" approach, which provides efficiency in processing large sets of structured data.
- On the other hand, this approach expands the range of applications to which the ability to work with objects is added.

*The strive for full joint use of the Database Approach and the Object-Oriented Approach is to build systems that are object-oriented and that have convenient query search facilities.*

# Questions and exercises:

1. Can you give an explanation of the word "object" ?
2. How do we call the process of grouping objects into classes ?
3. What is a basic concept in the object-oriented approach ?



# Thank you for your attention!



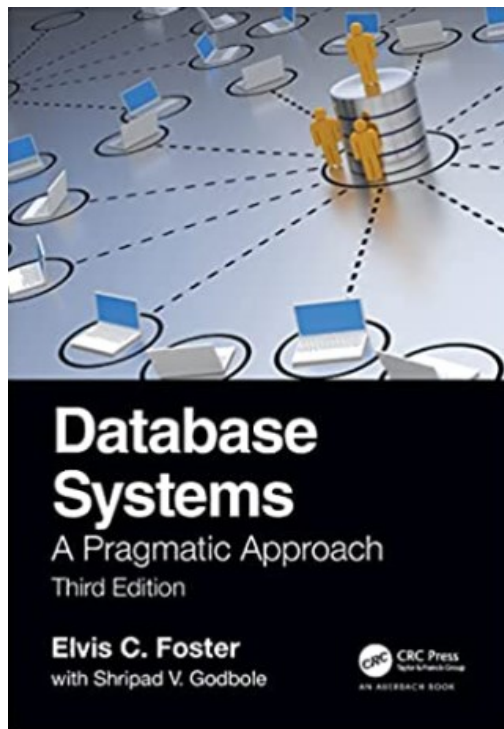
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 2. Relational approach in databases**
- ❑ **Topic 5. Object-oriented database systems.**
- ❑ Lesson 2. Object-oriented database management systems. Architecture.

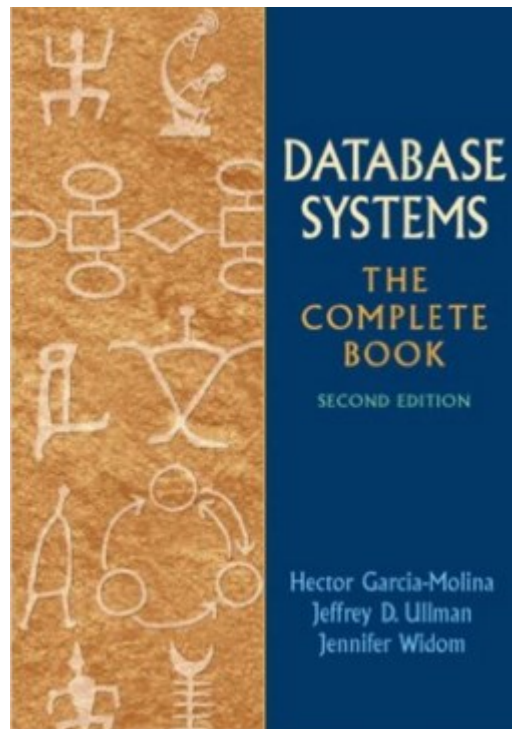


ERASMUS+

# OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS. ARCHITECTURE.



Elvis C. Foster, Shripad V. Godbole. (2022) Database Systems. A Pragmatic Approach (3rd edition), CRC Press.



Ullman, J., Widom, J. (2009) DATABASE SYSTEMS The Complete Book (2nd ed), Upper Saddle River, New Jersey.



Azalov, P. (1991) Database, Sofia.

# Object-oriented DBMS

- One of the areas of application of the **object-oriented** approach is databases.
- The main reason for this is the desire of specialists to expand the scope of applicability of **DBMS** in areas such as artificial intelligence, image and sound processing systems, office systems, and others.
- There are various reasons why modern **DBMSs** cannot be directly used for these purposes, but the most important of them can be summarized as follows:

*a) lack of capabilities to represent **complex data objects**;*

# Object-oriented DBMS

- b) lack of appropriate means of **expressing the behavior** of the modeled real objects;*
- c) lack of mechanisms **registering and processing** events that can occur with the objects of the modeled subject area.*

- There are two known approaches to overcome the shortcomings mentioned in the DBMS. Through one of them, called the "**bottom-up**" **approach**, the aim is to extend the traditional DBMS with an object-oriented concept and in the first place with the maintenance of complex information objects.

# Object-oriented DBMS

- The other, called the "**top-down**" **approach**, aims to extend object-oriented programming languages with the ability to maintain persistent objects, i.e. of objects that continue to exist and after the completion of the execution of the program.
- Both approaches have their advantages and disadvantages. However, there is a unanimous opinion that an object-oriented DBMS should be a "*symbiosis*" of the main ideas of these approaches. This gives reason to assume that a system is an object-oriented DBMS (OODBMS) if it satisfies two basic requirements:



# Object-oriented DBMS

1. *It is a DBMS in the traditional sense of the term.*
2. *It is object-oriented, i.e. supports object classes, inheritance and polymorphism, object identification and extensibility.*

## TYPES, CLASSES AND OBJECTS

The concept of **type** is fundamental in modern programming languages. A **type** unites a set of elements, called values, and a corresponding set of operations applied to those values.

- Two types of types are used - **simple** (atomic) and **composite** (structural). Composite types include both simple and composite types.

# Object-oriented DBMS

- In this way, it is possible to build a hierarchy of **types**. Constructors such as array, list, record, and set are used for this purpose.
- Each variable is considered as a quantity receiving values of a specific **type**, which is also assumed to be its **type**.
- Despite some common features, the concept of **class** is different from the concept of **type**. While **types** are mainly used during compilation when checking for correctness of the program, **classes** are designed to create and maintain objects.
- The **classes**, as well as the **types**, classify the values into separate sets.



# Object-oriented DBMS

- Unlike **types**, however, **classes** support another set (class extension), which includes the actually generated instances, called **objects**.
- Each **object** consists of a certain number of variables defining its structure. In some object-oriented systems, the type of variables can be specified at program execution time.
- This naturally creates greater flexibility, but requires a mechanism for the so-called late connection, due to the fact that it is not clear in advance which of the methods (procedures) will be used.

# Object-oriented DBMS

- **Objects** in object-oriented systems are encapsulated. This is expressed in the presence of two main parts in each object - an *interface* and an *realization (implementation) part*.
- The interface part represents a set of operations that can be applied to the **object**.
- The implementation part describes the structure of the **object** and the way in which each of the operations is implemented.
- **Objects** are accessible only through their interface part, while the implementation part is "hidden", i.e. inaccessible. This way, conditions are created to make changes to the implementation part without affecting the programs using the objects.

# Object-oriented DBMS

- Protection of application programs from changes in the implementation part is a property of object-oriented systems, called implementation independence or data independence.

## COMPLEX OBJECTS AND EXPANDABILITY

In the implementation of some non-standard applications, the basis of which the DBMS is used, there is a need to expand the system with capabilities both for defining **complex data objects** and for corresponding operations for their processing.

- **Complex or composite data objects** consist of other objects that can also be complex.

# Object-oriented DBMS

- The structure of the **complex object** is determined by the various constructors and by the sequence in which they are applied over the individual components (sub-objects)..
- The relational model does not allow defining and operating with **complex objects**. Constructors allowed in it are only record ( $n$ -tuple) and set, but their nesting is not allowed.
- Objects in object-oriented data models are **complex objects** without limitation in the structure of their constituent components.

# Object-oriented DBMS

- The use of **complex objects** requires the implementation of corresponding methods for operating on them. And since an object-oriented DBMS combines the capabilities of an object-oriented programming system and a classical DBMS, it can be assumed that these operations are provided by the programming environment.
- In such cases, however, it will be necessary to transform the data from the database into a form appropriate for the particular environment, which will consume too much time. As a consequence, complex object processing operations should represent a natural **extension** and complement of **DBMS** operations.

# Object-oriented DBMS

## IDENTIFICATION AND STABILITY OF OBJECTS

- In object oriented DBMS, objects are distinguishable from each other. Hence the requirement for their **identification**.
- While in programming languages the **identification** of objects is reduced to addressation and in the DBMS to the value of a primary key, the **identification** of objects in object-oriented DBMSs does not depend on their physical location, nor on the specific values of their attributes.
- In many systems, the problem of object **identification** is solved by systematically generating an **identifier** of each object, which uniquely identifies it throughout the object environment and is independent of its physical location.

# Object-oriented DBMS

- The **persistence** of some objects in object-oriented DBMSs is expressed in their ability to "exist" after another working "session" with the system, i.e. they can be repeatedly used.
- An important problem concerning the **sustainability** of objects is the definition of conditions under which some objects are **sustainable** and others are not.
- Different solutions to this problem are possible. It could be assumed that every created object becomes **sustainable**. However, this would mean that also the objects obtained as intermediate results would have this property.



# Object-oriented DBMS

- Another possibility is to specify **sustainability** at class level, i.e. only the objects of individual classes will be **sustainable**. In addition to this option, the possibility may be provided for objects of other classes to be specifically indicated as **sustainable**.
- A third possibility is that the **sustainability** of the object is in function of the operations with which it was obtained.

## QUERY LANGUAGES

- A common case in classical DBMS is that the actual processing of the data is described by means of a general-purpose programming language such as Fortran, Pascal, or C, and the access to the data from the database is described by means of a data sublanguage such as SQL.

# Object-oriented DBMS

- The above approach has significant disadvantages, and this is one of the considerations for object-oriented DBMS development.
- The requirements that are placed on query languages in object-oriented DBMSs are, for the most part, also requirements for classical DBMSs.
- A brief overview of the more significant ones follows below:

*1. The query language must be a high-level language. This means that regardless of the object type, which is generally a complex object, the query language must provide simple expressions for its processing.*

# Object-oriented DBMS

*2. The query language should be independent of specific applications, i.e., it must be universal in relation to the subject area under consideration.*

*3. Query language closure. In addition to object processing tools, the query language must provide the means to build and maintain class-hierarchy and heredity, define complex objects, manage the objects obtained as results of some operations, etc.*

# Object-oriented DBMS

*4. Query language power. In addition to the expressive power of relational languages, languages in object-oriented DBMS must allow the definition and processing of recursive objects, as well as navigation along the corresponding class-hierarchy.*

*5. **Query optimization**. The query language in the object-oriented DBMS must allow for opportunities for system optimization.*

# Architecture of object-oriented DBMS

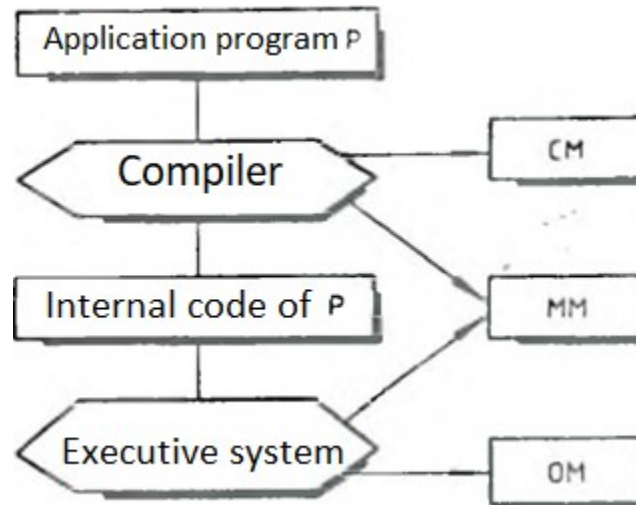
- The advantages that an object-oriented DBMS has over a classical DBMS give rise to new and difficult problems that must be overcome when creating such systems.
- Although there is still no unified concept of the architecture and functions of object-oriented DBMS, based on some of the already created DBMS, we will look at the main components that build them.

## BASIC COMPONENTS

The basic components of an object-oriented DBMS are several, and each of them can be viewed as a separate subsystem.

# Architecture of object-oriented DBMS

- Such are the manager of objects, cited earlier as **OM** (**object-manager**), **MM** the manager of methods (**method manager**) and **CM** - manager of classes (**class manager**).



Basic components of an object-oriented DBMS

# Architecture of object-oriented DBMS

- The most important functions of **OM** are to create and destroy, store and maintain the objects in the database. Activation of **OM** is done by sending messages, where **OM** knows the name of the message and the identifiers of the objects participating in it.
- In order to execute the specified method, the **OM** must first establish the belonging of the objects to their respective classes.



# Architecture of object-oriented DBMS

- The tasks of the **CM** are to create and store the class descriptions of objects and establish relationships between them, thus building the class-hierarchy of classes.
- Referred to the functions of a classic DBMS, the functions of the **CM** are close to those, up to the maintenance of the DBMS catalog.
- The purpose of **MM** is to create, store, and maintain methods in source and object code.
- The methods are system (built into the system itself) and user.

# Architecture of object-oriented DBMS

- Methods are defined by the user within the newly created classes needed for a particular application. User methods are written in a specific **object-oriented programming language**.
- It is difficult to unite in one the diversity of concepts underlying the architectures of **object-oriented DBMSs**. For this reason, only their characteristic components are presented here.
- Unlike object-oriented programming languages, in which objects exist only during the execution of the corresponding program, **object-oriented DBMSs** are characterized by the possibility of working with persistent objects.

# Architecture of object-oriented DBMS

- Each object is assumed to be uniquely identified by its own identifier, by which it is accessible both in RAM and external memory.
- In many implementations, this identifier is a pair of names - the name of a file and the name of a record containing the object. They can be encoded in four bytes.
- With **11 bits** for a file name and **21 bits** for a record name, the number of files can be up to **2048**, and the number of records in each of them can be over two million.
- Objects located in external memory have the same status as those in **RAM**. For this purpose, a virtual memory is maintained, as the objects that are present in the operating memory are registered in a special table.

# Architecture of object-oriented DBMS

- Each object is physically constructed from two parts. One of them is systemic and includes: information about the class to which the object belongs; a counter of the objects for which the object in question is subordinate, i.e. it is a subset of them and two flags specifying whether the object has undergone changes and whether to the object has been send commanded for its destruction.
- Objects that are  $n$ -tuples of other objects or represent sets of objects contain only primitive values or identifiers of their constituent objects.

# Architecture of object-oriented DBMS

- Because of the mutability of sets as structures, their representation in object space requires an appropriate memory allocation strategy as well as appropriate file system capabilities to handle files whose records are of variable or indefinite length.

*Despite the simplicity of adopting the relational approach and the use of relational schemas, some structural limitations are the reason for searching for new ways. One of them is the development of new data models with richer semantics. These are, for example, the Entity-Relationship data model, the Semantic Data Model, and the **Object-Oriented Data Model**.*

# Benefits of OODBMS

01.	<b>Reusability of Code:</b> A tested system component can be reused in the design of another component.
02.	<b>Stability and Reliability:</b> Software can be constructed from tested components. Organizations can be assured of guaranteed performance of software.
03.	<b>Increased Sophistication:</b> More complex systems can be constructed.
04.	<b>Understandability:</b> Designer and user think in terms of object and behavior rather than low-level functional details. This results in more realistic modeling that is easier to learn and communicate.
05.	<b>Faster Design:</b> Most RAD tools and contemporary CASE tools are object-oriented to some degree. Also, code reusability enhances faster development.
06.	<b>Higher Quality Design:</b> New software can be constructed by using tested and proven components.
07.	<b>Easier Maintenance:</b> Since systems are broken down into manageable component objects, isolation of system faults is easy.
08.	<b>Dynamic Lifecycle:</b> I-OO-CASE tools integrate all stages of the software development life cycle (SDLC).
09.	<b>Interoperability:</b> Generic classes may be designed for multiple systems.
10.	<b>Design Independence:</b> Classes may be designed to operate and/or communicate across different platforms.
11.	<b>Clarity:</b> OT promotes better communication between IS professionals and business people.
12.	<b>Better CASE Tools:</b> OT leads to the development of more sophisticated and flexible CASE and RAD tools.
13.	<b>Better Machine Performance:</b> OT leads to more efficient use of machine resources.

## Questions and exercises:

1. Which of the following is true concerning an OODBMS?
  - A. They have the ability to store complex data types on the Web.
  - B. They are overtaking RDBMS for all applications.
2. Which of the following is an ordered collection of elements of the same type ?
  - A. Set
  - B. Bag
  - C. List
  - D. Dictionary





# Thank you for your attention!

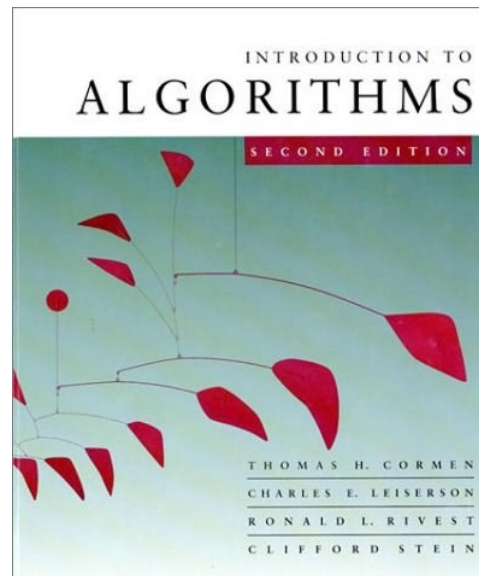
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 1. Introduction to algorithms**
  - ❑ Lesson 1. The role of algorithms in computation.  
Algorithms as a technology.



ERASMUS+

# THE ROLE OF ALGORITHMS IN COMPUTATION. ALGORITHMS AS A TECHNOLOGY.



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009).  
Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.

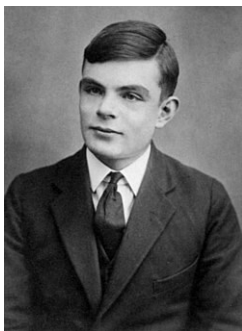
# Introduction

- What are algorithms?
- Why studying algorithms is useful?
- What is the role of algorithms used in computers?
- In this lecture, we will try to give answers to these questions.

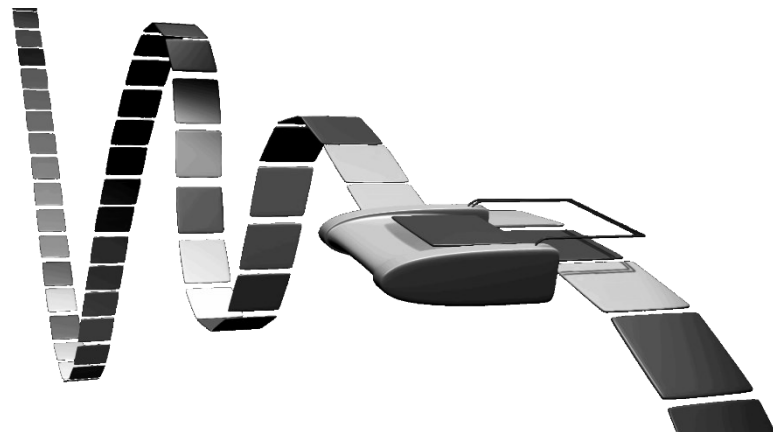
# Algorithms

- **Algorithm** (from the name of the scholar Al-Khawarizmi) is a term from mathematics, computer science, linguistics, and other fields that refers to a finite sequence of instructions or an explicit description of a step procedure used for solving a problem, often related to computation or data processing.

Alan Mathison Turing



Turing machine



ERASMUS+

# Algorithms

Turing machine is an imaginary computer described by the English mathematician **Alan Turing** in 1936. Turing's work is the first accurate definition of the notion algorithm (also called mechanical, formal or effective procedure). It is used to prove basic results in computer sciences mainly in the fields calculation and complexity of algorithms as well as mathematical logic.

**Turing machine** consists of four components:

1. **Memory** – potentially infinite tape consisting of cells, in each cell is written a symbol of some finite alphabet. The alphabet contains a special blank symbol (usually written as “0”) and one or more other symbols. Every moment of machine's work the tape is finite, but if needed we can add new cells on the left or on the right, containing the blank symbol.
2. **Head** that in each moment of calculation is over a definite cell of the tape. In each tact the head reads the symbol of the cell over which is situated, writes a new symbol and moves left or right on the tape depending on the instruction and the read symbol.
3. **Program** - a finite list of instructions which unlike contemporary computers is separate from the memory. Each instruction is a set of indications what to be done if the head has read the i-th letter from the alphabet. Each indication contains information which symbol should be written back on the tape, which instruction will be performed on the next step and where (left or right) to move the head.
4. **Register** containing the number of the active instruction (program counter). One instruction is said to be initial i.e. calculation starts with entering its number in the program counter. There is a final instruction as well – when reaching it the calculation stops.

# Algorithms

- Informal, an algorithm is any well-defined computational procedure that takes one or more values as input and exports another value as output.
- In other words, algorithms are like recipes for performing a well-defined task.
- In this line, some block code that computes, for example, the Fibonacci number sequence is an execution of a particular algorithm.
- Even a simple function for addition two numbers can be considered an algorithm, even a very simple one.



# Algorithms

- Algorithms should be studied first because they do not depend on a particular technology.
- You don't need to know Java or C++ or any new technology to understand algorithms.
- It requires having an understanding towards the concept of the steps that should be followed to solve a certain problem (with certain programming languages with similar features).
- Algorithms are important for the computer sciences field because they enable making analysis of different calculation ways to achieve the best way for solving a problem.

# Algorithms

- By "**best way**" is meant the one that with the least **resources** can get to the solution the fastest.
- An algorithm is a powerful tool for solving a well-defined computational problem.
- For example, we may need to sort a sequence of numbers in ascending order.
- This problem arises frequently in practice and gives good grounds for the introduction of many standard techniques and analysis tools.

# Algorithms

- Here's how we can define the sorting problem:
- **Input:** a sequence of  $n$  numbers:  $\langle a_1, a_2, \dots, a_n \rangle$
- **Output:** change (reordering)  $\langle a'_1, a'_2, \dots, a'_n \rangle$ , so that  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

## Example:

**Input:** 8 2 4 9 3 6

**Output:** 2 3 4 6 8 9

# Algorithms

- Such an input sequence is called an instance of the sorting problem.
- In general, an instance of a problem consists of the input (satisfying constraints imposed for the problem) needed to calculate the problem solution.
- Because many programs use it as an intermediate step, sorting is a fundamental operation in computer sciences.
- As a result we have a large number of good sorting algorithms that we can use.

# Algorithms

- Which is the best algorithm to perform a particular activity depends more on other factors, such as:
  - the number of items to be sorted
  - possible restrictions on the values of the elements
  - Architecture of the computer
  - the type of storage devices: main memory, disks, etc.
- An algorithm is said to be correct if for every correct input, it stops with the correct output.
- We say that the correct algorithm solves a given computational problem.

# Algorithms

- The incorrect algorithm may not stop at all in some input cases, or may stop with an incorrect answer.
- Contrary to what might be expected, incorrect algorithms can sometimes be useful if we can control their error rate.
- Usually, however, we should strive to use and compile correct algorithms.
- An algorithm can be defined as a computer program, as long as an accurate description of the computational procedure to be followed is provided.

# Algorithms

- What problems are solved by algorithms?
- Practical applications of algorithms are everywhere and include the following examples
  - Algorithms for defining gene sequencing in bioinformatics
  - The Internet enables people around the world to quickly access and retrieve large amounts of this information. Using clever algorithms, Internet objects are able to manage and manipulate this large volume of data.



# Algorithms

- What problems are solved by algorithms?
- Practical applications of algorithms are everywhere and include the following examples:
  - E-commerce enables goods and services to be contracted and exchanged electronically, and it depends on the privacy of personal information such as credit card numbers, passwords and bank statements.
  - Technologies used in e-commerce include cryptography and digital signatures, which are based on numerical algorithms and the number theory.

# Algorithms

- What problems are solved by algorithms?
- Manufacturing and other commercial enterprises often need to allocate scarce resources in the most useful way.
- A petrol company may wish to know where to place its wells in order to maximize expected profits.
- A political candidate may want to determine where to spend money on an advertising campaign in order to increase the chances to win the elections.
- An Internet service provider may want to determine where to place additional resources to serve its customers more effectively.

# Algorithms

- What problems are solved by algorithms?
- **These are all examples of problems that can be solved using linear programming, which we will study in later lectures.**
- Some of the most important algorithms a computer engineer or programmer should know:
  - Binary search – a technique for finding a particular value in a one-dimensional array (or in any data structure that is sorted) by excluding half of the data at each step.
  - Dijkstra's algorithm – Dijkstra's algorithm is used to find the shortest path between the nodes of a graph, and they can represent road network.
  - Floyd-Warshall – The Floyd-Warshall algorithm is used to find the shortest paths between all pairs of vertices in a weighted oriented graph.
  - Kruskal algorithm – type of greedy algorithm. This is any algorithm that is implemented by selecting (locally) the maximum (or minimum) element of a set at each step, and the goal is to reach the global maximum.

# Algorithms as a technology

- Suppose that computers are infinitely fast and the computer memory is infinitely large. Is there a reason to learn algorithms?
- The answer is yes, if for no other reason than to show that our algorithm terminates after a certain time and finds the correct answer.
- If computers were infinitely fast, any proper problem solving method would work.
- You'll probably want your implementation to be among the best in software engineering practice (e. g., your implementation should be well designed and documented).

# Algorithms as a technology

- Computer time and memory are therefore limited.
- You need to use these resources wisely, and using algorithms that are time or memory efficient will help you do this.
- Different algorithms developed to solve the same problem often differ completely in their effectiveness.

# Algorithms as a technology

- In general, system performance depends on both the choice of fast hardware and the use of efficient algorithms.
- Algorithms are the basis of most technologies used in modern computers.
- Furthermore, with the ever-increasing capabilities of computers, we are using them to solve bigger problems.
- Having a solid knowledge base and algorithmic technique is one characteristic that separates truly skilled programmers and engineers.

# Algorithms properties

## Input data

- In the description of each algorithm, quantities are used. Some of them are constants and others are variables.
- Before being used, the variables must be given values called initial values.
- Some of the variables get their initial values within the description of the algorithm itself. Other variables receive their initial values as input data, i. e., during the execution of the algorithm.



# Algorithms properties

## Input data

- This way the variables act as parameters of the algorithm.
- For example, in Euclidean's algorithm they are the natural numbers that the variables  $M$  and  $N$  take as values.
- There are cases where the set of inputs of some algorithms is the empty set.

# Algorithms properties

## Output data

- The execution of each algorithm is aimed at obtaining a certain result.
- Most often, this result is an extreme value of some of the quantities used in the algorithm.
- These final values depend on the initial values, i. e. on the input data.
- The output result in Euclidean's algorithm is the extreme value of the variable  $a$  (or  $b$ ).

# Algorithms properties

## **Algorithms have the following important properties:**

1. **Massiveness**– the algorithm is used to solve a class of homogeneous tasks, not individual specific tasks. For example, Euclidean's algorithm allows to find the greatest common divisor of any two natural numbers. The effectiveness of the application of algorithms is due to their massiveness. Thanks to it an algorithm can be repeatedly applied in different cases. Each algorithm can be applied to a certain class of data, which we will refer to as the algorithm's eligibility.

# Algorithms properties

**Algorithms have the following important properties:**

2. **Definiteness** – the description of the algorithm is clear and contains no ambiguities. It defines unambiguously the actions to be carried out. Given certain data (values of the external variables), the algorithm produces the same result regardless of who the executor is. Definiteness allows algorithms to be executed by a wide range of implementors. Each of the instructions in the algorithms must be written in a way that avoids ambiguity.

# Algorithms properties

## **Algorithms have the following important properties:**

3. **Effectiveness** - a guarantee that, given any admissible values of the initial data, a finite number of elementary operations will lead to the desired result. Finalizing the algorithm in finite time. This property of algorithms allows them to be practically applied. The finite execution time of an algorithm is a consequence of the assumption that each algorithm consists of a finite number of steps and that each step (each elementary action) is executed in a finite time.

# Algorithms properties

**Algorithms have the following important properties:**

4. **Discreetness** - the algorithm consists of a finite number of elementary prescriptions. Very often, instead of "elementary prescription", other terms with the same meaning are used - instruction, command, operation or operator.

***The algorithm is a procedure for solving a mass task over given objects, which determinately, in a finite number of steps, starting from specific input data and applying the allowed operations leads to the desired result.***

# Conclusion

- The role of algorithms in everyday life as well as in science and engineering is too great.
- Algorithms are an appropriate and convenient form for the exposition of scientific results - there are a number of journals in which reports of various newly discovered methods, mainly in the field of mathematics, are printed in the form of algorithms.
- The great importance of algorithms determines the interest in them. Experts from every field of science and technology are looking for algorithms to solve various tasks both for immediate application in practice and for the development of science.
- **Algorithms are widely used in database query optimization (the process of choosing a suitable execution strategy for processing a query).**



# Questions and exercises:

1. What are algorithms and why studying algorithms is useful?
2. What is the role of algorithms in computation ?
3. What are the main properties of algorithms?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

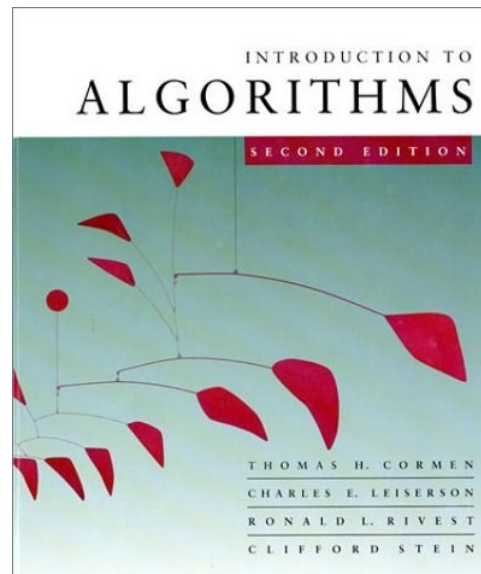
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 1. Introduction to algorithms**
  - ❑ Lesson 2. Design and analysis of algorithms



ERASMUS+

# DESIGN AND ANALYSIS OF ALGORITHMS



[Carmen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009). Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.

# Analysis of algorithms

- Analyzing an algorithm comes from predicting the resources that the algorithm requires.
- Sometimes, resources such as memory, or computer hardware are essential, but most often we want to determine the **computation time**.
- In general, when analyzing several candidate algorithms to solve a problem, we need to identify the most efficient one.
- Such an analysis may show more than one candidate suitable for solving the problem, but we may throw out some algorithms in the process of analyzing.

# Analysis of algorithms

## Analysis of sorting by insertion

6 5 3 1 8 7 2 4



The time required for the insertion sorting procedure depends primarily on the size of the input: sorting a thousand numbers takes longer than sorting three numbers.

- Moreover, sorting by insertion may have different times to sort two input sequences of the same size depending on how they are already sorted.
- In general, the running time of the algorithm grows with the size of the input, so traditionally the analysis consists in describing the **running time of the algorithm as a function of the size of its input.**

ERASMUS+

# Analysis of algorithms

## Analysis of sorting by insertion

- To do this, we need to define the terms "**runtime**" and "**input size**" more carefully.
- The idea of the size of the input depends on the problem being examined.
- For many problems, such as sorting, the most natural measure is the number of elements per input, e.g. the size  $n$  of a given sort array.



# Analysis of algorithms

## Analysis of sorting by insertion

- For many other problems, such as multiplying two integers, the best measure of input size is the **total number of bits** needed to represent the input in binary format.

Multiplicand  
Multiplier

$$\begin{array}{r} 1100_2 = 12 \\ \times 1101_2 = 13 \\ \hline 1100 \\ 0000 \\ 1100 \\ 1100 \\ \hline 10011100_2 = 156 \end{array}$$

Binary multiplication is easy  
 $0 \times \text{multiplicand} = 0$   
 $1 \times \text{multiplicand} = \text{multiplicand}$

Product

$$10011100_2 = 156$$

m-bit multiplicand  $\times$  n-bit multiplier = (m+n)-bit product

- Sometimes it is more appropriate to describe the input size with **two numbers** rather than one.
- For example, if the algorithm input is **related to a graph**, the size of the input can be described by the **number of vertices and edges of the graph**.
- We should specify which input measure size will be used for each problem that we're studying.

# Analysis of algorithms

## Analysis of sorting by insertion

- The running time of an algorithm for a particular task is the number of primitive operations or "**steps**" that are executed.
- It is convenient to define the notion of **step** so that it is **machine independent**.
- If we assume that each step of a source code takes a constant time to execute, then we will have a different time required for each step.
- This point of view is consistent with the RAM model, and it also reflects how **pseudo-code** would be implemented on the most computers.

# Analysis of algorithms

## Analysis of sorting by insertion

- In our analysis of sorting by insertion, it is necessary to consider **both: best case** in which the array elements are already sorted, and the **worst case** in which the input array is reverse sorted.
- The **worst-case** timing of the algorithm gives us an **upper bound** on the running time for each input.
- Knowing this ensures that the algorithm will never take longer.

# Analysis of algorithms

## Analysis of sorting by insertion

- For some algorithms, the worst case happens quite often.
- For example, when **searching a database** for a particular piece of information, the search algorithm will fall into the worst case where the information is not available in the database.
- In some applications, searching for missing information is quite common.



# Analysis of algorithms

## Pseudo-code

- Pseudo-code is a program code that is unrelated to a computer's hardware and requires rewriting the code for a computer before the program can be used.
- There are no strict syntax rules, it's designed for humans, not computers.
  - **High-level** description of the algorithm
  - **More structured** than human language
  - **With less details** than the program
  - **Preferred notation** for describing algorithms
  - **Hides problems** when writing a program.

# Analysis of algorithms

**Example: Finding the largest element of an array.**

**Algorithm** arrayMax(A, n)

**Input:** array A of n integers

**Output:** maximum element of A

currentMax  $\leftarrow$  A[0]

**for** i  $\leftarrow$  1 **to** n-1 **do**

**if** A[i] > currentMax **then** currentMax  $\leftarrow$  A[i]

**return** currentMax

# Analysis of algorithms

## Pseudo-code details

- Managing operators

**if...then...[else...]**

**while...do...**

**repeat...until...**

**for...do...**

The space replaces the brackets.

- Method defining

Algorithm method(arg[, arg...])

Input...

Output...



# Analysis of algorithms

## Pseudo-code details

- Calling a function/method  
**var.method (arg[, arg...])**
- Value return  
**return** expression
- Expressions
  - <- Assignment (like = in C++)
  - = Equality testing (like == in C++)
  - $n^2$  - degrees, etc. mathematical notations are allowed.

# Analysis of algorithms

## Elementary operations (primitives)

- Expression calculation.
- Assigning a value to a variable.
- Indexing in an array.
- Calling a function.
- Return a value from a function.

# Analysis of algorithms

## Counting the elementary operations

By checking the pseudo-code, we can determine the maximum number of primitive operations performed by the algorithm as a function of the input size.

### Example

Algorithm <code>arrayMax(A, n)</code>	Number of operations
<code>currentMax &lt;- A[0]</code>	2
<b>for</b> <code>i &lt;- 1 to n-1 do</code>	$2 + n$
<b>if</b> <code>A[i] &gt; currentMax then</code>	$2(n - 1)$
<code>currentMax &lt;- A[i]</code>	$2(n - 1)$
{increment counter <code>i</code> }	$2(n - 1)$
<b>return</b> <code>currentMax</code>	1
	$7n - 1$

# Analysis of algorithms

## Estimation of running time

- The arrayMax algorithm performs  $7n-1$  elementary operations in the worst case.

- Define:

$a$  = the running time of the fastest elementary operation.

$b$  = the running time of the slowest elementary operation.

- Let  $T(n)$  be the worst case for arrayMax. Then  $a(7n - 1) \leq T(n) \leq b(7n - 1)$ .
- The running time  $T(n)$  is therefore bounded by two linear functions.

# Design of algorithms

- Many useful algorithms are recursive in structure: to solve a given problem, they **recursively** call upon themselves one or more times to deal with closely related subproblems.
- These algorithms typically follow a “**divide-and-conquer**” approach: they divide the problem into several subproblems that are similar to the original problem but smaller in size, and solve the subproblems recursively, and then combine solutions to create a solution to the initial problem.

# Design of algorithms

- The divide-and-conquer paradigm involves three steps at each level of recursion:
  - **Dividing** the problem into a number of sub-problems that are smaller copies of the main problem;
  - **Conquering** the sub-problems and solving them recursively. If the sub-problem sizes are small enough, however, just solve the sub-problems in a simple way.
  - **Combining** the solutions to the sub-problems into a solution for the initial problem.

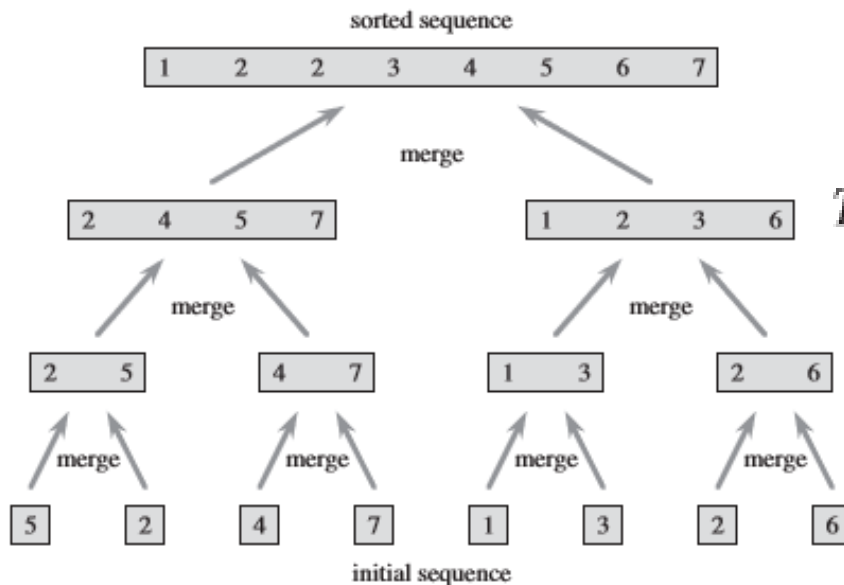
The algorithm for **sorting by merging (merging sort)** uses the divide and conquer paradigm.

# Design of algorithms

- When an algorithm contains a recursive calling itself, we can often describe the running time with a recurrence or iteration equation that describes the running time on a problem of size  $n$  in terms of the running time on smaller inputs.
- We can then use mathematical tools to solve the iteration and provide bounds on the performance of the algorithm.



# Design of algorithms



$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

**divide-  
and-  
conquer**

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

**merge  
sort**

# Summary

- **The analysis part** includes the concepts of input size, **time complexity** and asymptotic growth rate bounds.
- Except the worst-case complexity, analyzing of **average complexity** is also affected.
- **Techniques** for estimating the complexity of algorithms and the use of these techniques on classical samples are discussed.
- Different algorithms like time complexity and memory complexity are **analyzed**.

# Summary

- The **"design"** part introduces a number of algorithmic schemes designed to facilitate the creation of efficient algorithms: **"divide and conquer"**, **"dynamic programming"**, **"greedy"**, **"graph traversal"**, **"backtracking"** and many other algorithmic techniques and approaches.
- Schemes are applied to solve typical tasks.
- Schemes for compiling **approximation** algorithms are demonstrated.

# Questions and exercises:

1. What is analysis of algorithms?
2. What is design of algorithms?
3. What is pseudo-code?



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 3. Algorithms and their applications in databases for query optimization

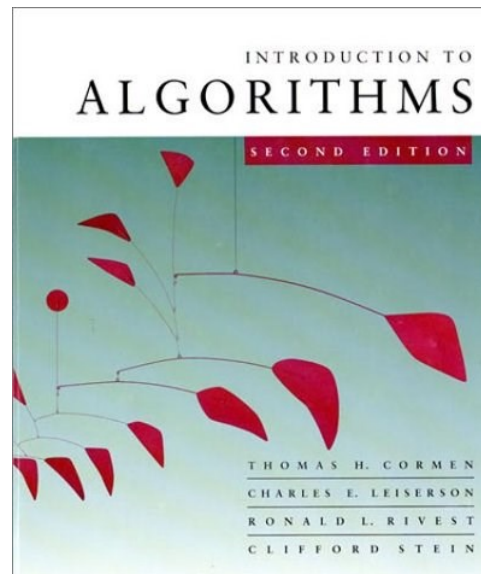
### ❑ Topic 1. Introduction to algorithms

- ❑ Lesson 3. Complexity of algorithms. Types of complexity and their estimation



ERASMUS+

# COMPLEXITY OF ALGORITHMS. TYPES OF COMPLEXITY AND THEIR EVALUATION.



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009). Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.



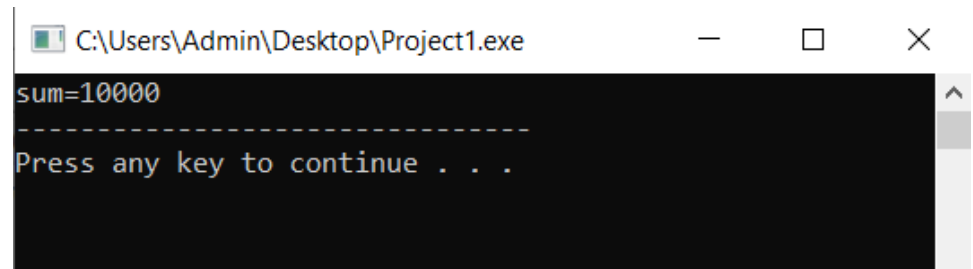
# Introduction

- The topic of algorithm evaluation and complexity is important and should not be overlooked.
- When considering a computer algorithm, we are generally interested in three of its properties:
  - **Simplicity** (and elegance) ;
  - **Correctness**;
  - **Fastness**.
- While the first of these anyone can "measure" intuitively (and somewhat subjectively), the last two require a much **deeper analysis**.

# Techniques for analyzing algorithms

- *Consider* the following program fragment of an algorithm:

- 1)  $n = 100$ ;
- 2)  $\text{sum} = 0$ ;
- 3) for ( $i = 0$ ;  $i < n$ ;  $i++$ )
- 4) for ( $j = 0$ ;  $j < n$ ;  $j++$ )
- 5)  $\text{sum}++$ ;

A screenshot of a Windows command prompt window titled 'C:\Users\Admin\Desktop\Project1.exe'. The window has standard Windows window controls (minimize, maximize, close). The command prompt shows the text 'sum=10000' on the first line, followed by a dashed line '-----' on the second line, and 'Press any key to continue . . .' on the third line. The cursor is positioned at the end of the third line.

We are interested in how fast the above algorithm will run. What we can do experimentally is to check how long will it take to finish its work.

# Techniques for analyzing algorithms

- To examine its behavior more generally, we can run it for other values of  $n$ .

Dependency between input data size and execution speed

Input size	Running time
10	0,000001 sec.
100	0,0001 sec.
1000	0,01 sec.
10000	1,071 sec.
100000	106,543 sec.
1000000	10663,6 sec.

- when we increase  $n$  ten times, the program execution time increases 100 times.

# Techniques for analyzing algorithms

- Let us examine this fragment more deeply

```
1)      n = 100;  
2)      sum = 0;  
3)      for (i = 0; i < n; i++)  
4)          for (j = 0; j < n; j++)  
5)              sum++;
```

- On lines 1) and 2) there is a static initialization that takes constant time. Let's denote it by  $a$ .
- For the operations  $i = 0$  and  $i++$ , as well as for the check  $i < n$ , again a constant time is required (each of them represents a constant number of instructions by the processor), we will denote it by  $b$ ,  $c$ ,  $d$ .
- In line 4) we denote the times required for the operations  $j = 0$ ,  $j < n$  and  $j++$  by  $e$ ,  $f$ ,  $g$ .
- Last, the operation on line 5) also requires constant time: let it be  $h$ .

# Techniques for analyzing algorithms

- With thus introduced notations, it is not difficult to calculate the total running time of the algorithm for an arbitrary value of  $n$ :

$$\begin{aligned} & a + b + n.c + n.d + n.(e + n.f + n.g + n.h) = \\ & = a + b + n.c + n.d + n.e + n.n.f + n.n.g + n.n.h = \\ & = n^2.(f + g + h) + n.(c + d + e) + a + b \end{aligned}$$

Remind that  $a, b, c, d, e, f, g, h$  are constants. Let denote:

$$i = f + g + h$$

$$j = c + d + e$$

$$k = a + b$$

(here  $i$  and  $j$  have nothing in common with the variables used in the fragment above). Thus the algorithm is running for time:

$$i.n^2 + j.n + k$$

# Techniques for analyzing algorithms

- The constants  $i$ ,  $j$  and  $k$  are important for the algorithm's fastness, but are not determinant. In practice, when we study the effectiveness of an algorithm, we are not interested in them. These constants depend primarily on the machine representation of our program, as well as the speed of the machine on which it is performed.
- Moreover, when examining the behavior of our algorithm, we can ignore even the single-valued  $j$ .  $n$  and  $k$  and keep only the one in which  $n$  participates to the most.
- The purpose of this "simplification" is to leave only the *most significant* feature for a given algorithm, i. e. , the feature on which the runtime depends the most, i. e. , which grows the fastest as the input data size increases.

# Techniques for analyzing algorithms

- Let consider two functions that show the running time of two given algorithms  $A1$  and  $A2$  depending on the size  $n$  of the input data:  $f = 2 \cdot n^2$  and  $g = 200 \cdot n$ .
- It is easy to see that although the coefficient of  $g$  is much larger than that of  $f$ , when  $n$  passes some fixed value (in this case  $n > 100$ ), the  $A2$  algorithm will solve the task faster than  $A1$ .
- Moreover, as  $n$  increases, the ratio between the running times of the two algorithms increases in favor of  $A2$ .
- Asymptotically, the  $A2$  algorithm is faster and its complexity is linear, while that of  $A1$  is quadratic.



# Techniques for analyzing algorithms

- **What is a complexity?**
- Briefly complexity is how the required time or memory to execute an algorithm changes by changing the input data size.
- In formally evaluating the complexity of algorithms, we will be interested in their behavior at  $n$  tending to infinity. We will describe the complexity of an algorithm by functions of the type  $f : N \rightarrow N$ .

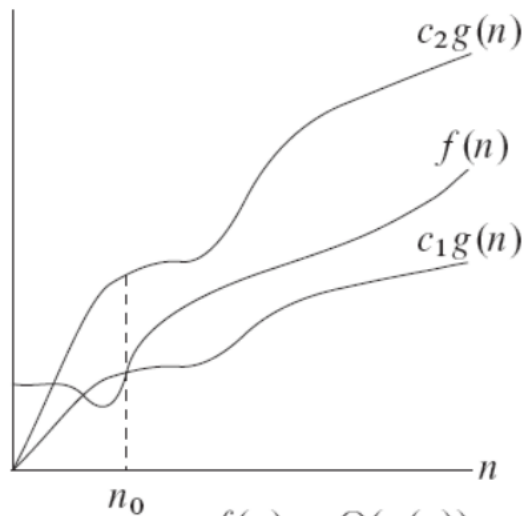
**Definition 1.27.**  $O(F(n)) = \{f(n) \mid \exists c (c > 0), \exists n_0(c): \forall n > n_0 : 0 \leq f(n) \leq c \cdot F(n)\}$

I.e.  $O(F(n))$  is a set of  $f$  functions for which such constant  $c$  ( $c > 0$ ) that  $f(n) \leq c \cdot F(n)$  exists for all *big enough* values of  $n$ , i.e. constant  $n_0$  exists (eventually depending on  $c$ ), for which the above inequation is satisfied for each  $n > n_0$ . In this way  $O(F)$  defines the set of all functions that *increase not faster* than  $F$ .

# Techniques for analyzing algorithms

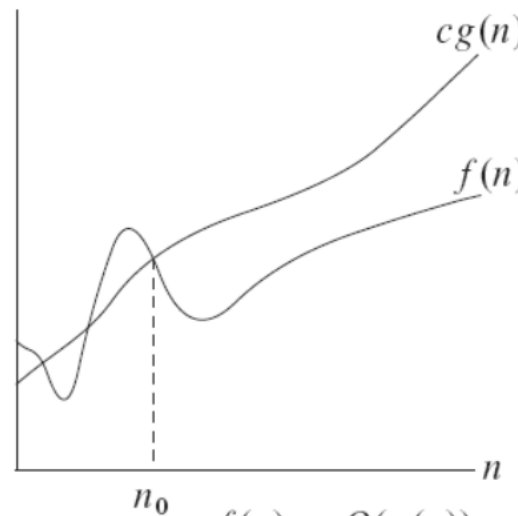
- **Asymptotic notation**
- 1.  **$O(F)$**  defines the set of all functions  $f$  that increase **no** faster than  $F$ , i. e. there exists a constant  $c > 0$  such that  $f(n) \leq cF(n)$ , for all sufficiently large values of  $n$ .
- 2.  **$\Theta(F)$**  defines the set of all functions  $f$  that increase **as fast as**  $F$  (with an allowance to a constant multiplier), i. e. there exist constants  $c_1 > 0$  and  $c_2 > 0$  such that  $c_1F(n) \leq f(n) \leq c_2F(n)$ , for all sufficiently large values of  $n$ .
- 3.  **$\Omega(F)$**  defines the set of all functions  $f$  that increase **no** slower than  $F$ , i. e. there exists a constant  $c > 0$  such that  $f(n) \geq cF(n)$ , for all sufficiently large values of  $n$ .

# Techniques for analyzing algorithms



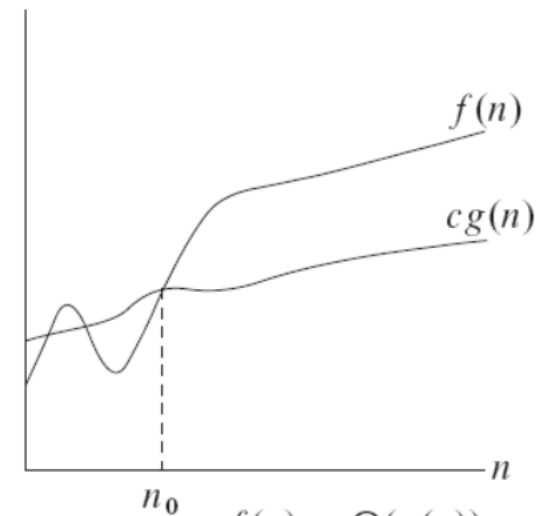
$$f(n) = \Theta(g(n))$$

(a)



$$f(n) = O(g(n))$$

(b)



$$f(n) = \Omega(g(n))$$

(c)

the complexity of an algorithm increases with the growth rate of the given function

The worst case

The best case

# Techniques for analyzing algorithms

- The complexity of an elementary operation is constant, i. e.  $O(1)$ . It is not easy to define what an elementary operation is. Basically, an elementary operation is one that is executed in a constant time, independent of the size of the data being processed. Elementary operations in the general case are for example assignment, addition, multiplication, etc.
- However, when working with hundred-digit numbers, it is not a good idea to consider multiplication an elementary operation. It is not good to consider trigonometric functions (sine, cosine, etc. ), exponent, logarithm as elementary operations.

# Techniques for analyzing algorithms

## *Sequence of operators*

The time complexity of a sequence of operators is determined by the complexity of the slower of them.

- Formally, if the operator  $s_1$  with complexity  $F_1$  is followed by the operator  $s_2$  with complexity  $F_2$ , we can write:

$$T(s_1) \in O(F_1), T(s_2) \in O(F_2) \Rightarrow T(s_1; s_2) \in O(\max(F_1, F_2))$$

This is equivalent to the rule:

$$f_1 + f_2 \in (\max(f_1, f_2))$$

# Techniques for analyzing algorithms

## *Composition of operators*

- When nesting an operator in the scope of another operator, the complexity is calculated as the product of their complexities, i. e.

$$T(s_1) \in O(F_1), T(s_2) \in O(F_2) \Rightarrow T(s_1\{s_2\}) \in O(F_1.F_2))$$

This is equivalent to the rule:

$$f_1.f_2 \in O(f_1.f_2)$$

# Techniques for analyzing algorithms

## *if-constructions*

```
if (p)
    s1;
else
    s2;
```

- If the complexities of  $p$ ,  $s1$  and  $s2$  are  $O(P)$ ,  $O(F1)$ ,  $O(F2)$ , then the complexity of the fragment shown is  $\max(O(P), O(F1), O(F2))$ , i. e. , the complexity of the fastest growing function among  $P$ ,  $F1$  and  $F2$ .



# Techniques for analyzing algorithms

## *Loops*

Let's look at the loop:

```
fact = 1;
for (i = 1; i <= n; i++)
    fact *= i;
```

- We can consider that the body of the loop takes a constant time  $c$  independent of  $n$ .
- The complexity of the for loop operator is  $O(n)$ . Then by the composition rule for the complexity of the whole loop we get  $O(c \cdot n)$ , i. e.  $O(n)$ .
- Here we should add the complexity of the initial initialization before the loop (which has complexity  $O(1)$ ), where, by the consistency rule, we get:  $O(1+n)$ . At the end, the complexity turns out to be  $O(n)$ .

# Techniques for analyzing algorithms

## *Nested loops*

```
sum = 0;  
for (i = 0; i < n; i++)  
    for (j = 0; j < n; j++)  
        sum++;
```

- The complexity of two or more nested loops with mutually independent counters can be easily derived. In the case of two nested loops from the fragment above, it is  $f \in n \cdot O(g)$ , where  $g$  is the complexity of the inner loop. But  $g \in O(n)$ , then  $f \in O(n \cdot n)$ , i. e.  $f \in O(n^2)$ .

# Techniques for analyzing algorithms

## *Example*

```
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        if (i == j)
            for (k = 0; k < n; k++)
                sum++;
```

- The *if* operator will execute  $n^2$  times, but only  $n$  times the result of the  $i == j$  check will be true. Since the complexity of the innermost loop is linear, we obtain a total complexity of  $O(n^2)$ .

# Techniques for analyzing algorithms

## *Logarithmic complexity*

Let's look at the program fragment:

```
for(sum = 0, h = 1; h < n; h *= 2)
    sum++;
```

- Here  $h$  takes values  $1, 2, 4, \dots, 2^k, \dots$  until it reaches  $n$ . Thus  $\text{sum}++$  is executed  $\log_2 n$  times and the complexity of the algorithm is  $O(\log_2 n)$ .
- Such algorithmic complexity has a binary search as well, where at each step the search interval is divided into two (almost) equal parts.

# Techniques for analyzing algorithms

## ***Recursion***

- Analyzing recursion in the general case is not trivial. Typically, a dependence of the form  $T(n) = f(T(n-1))$  is obtained for the complexity of the algorithm.

### ***- Factorial***

Consider the function:

```
unsigned fact(unsigned n)
{ if (n < 2)
    return 1;
  return n*fact(n-1);
}
```

In this case the recursion is equivalent to a single loop of type *for*, where for the complexity we easily get  $O(n)$ .

# Techniques for analyzing algorithms

## *Recursion*

### - *Fibonacci numbers*

However, things are not always so simple. Take Fibonacci numbers for example, which can be found extremely inefficiently with recursion:

```
unsigned fib(unsigned n)
{
    if (n < 2)
        return 1;
    return fib(n-1) + fib(n-2);
}
```

For  $n = 0$  and  $n = 1$ , we have constant time complexity: one elementary check and return a result. In the other case we have the same check again, but this time followed by two recursive conversions.

# Techniques for analyzing algorithms

## *Recursion*

### - *Fibonacci numbers*

In general, the following formulas are valid:

$$\begin{aligned}T(0) &= T(1) = O(1) \\T(n) &= T(n-1) + T(n-2) + O(1), n \geq 2\end{aligned}$$

The above dependencies strongly remind of these from the Fibonacci numbers definition:

$$\begin{aligned}f_0 &= f_1 = 1 \\f_n &= f_{n-1} + f_{n-2}\end{aligned}$$

It follows immediately that  $T(n) \geq f_n$ . However for the Fibonacci numbers the following independencies are valid:  $\left(\frac{3}{2}\right)^{n-1} \leq f_n \leq 2^n, n \geq 1$  (It is proved slightly by induction). So it turns out that  $T(n)$  increases exponentially.



# Techniques for analyzing algorithms

## ***Recursion***

### ***- Fibonacci numbers***

However, if we are smart enough not to recompute something we have already computed, we can reduce the complexity of the algorithm to  $O(n)$ :

```
unsigned long f[MAX] = {0,0,0,...};  
unsigned long fib(unsigned n)  
{  
    if (0 == f[n])  
        if (n < 2)  
            f[n] = 1;  
        else  
            f[n] = fib(n-1) + fib(n-2);  
    return f[n];  
}
```

# Typical complexities of algorithms

Complexity	Sign	Description
constant	$O(1)$	A constant number of steps (say 1, 5, 10, or some other number) is required to perform an operation, and this number is independent of the input data volume.
logarithmic	$O(\log(N))$	To perform an operation on $N$ elements requires a number of steps of the order of $\log(N)$ , where the base of the logarithm is usually 2. For example, an algorithm with complexity $O(\log(N))$ for $N = 1,000,000$ will take about 20 steps (to constant precision).
linear	$O(N)$	To perform an operation on $N$ elements requires approximately as many steps as there are elements. Approximately 1,000 steps are needed for 1,000 items. The number of elements and the number of operations are linearly dependent, for example the number of steps is about $N/2$ or $3*N$ for $N$ elements.
Log-Linear	$O(n*\log(n))$	It takes approximately $N*\log(N)$ steps to perform an operation on $N$ elements. Approximately 10,000 steps are needed for 1,000 elements.

# Typical complexities of algorithms

Complexity	Sign	Description
quadratic	$O(n^2)$	An operation requires $N^2$ number of steps, where $N$ characterizes the input data volume. For example, an operation on 100 items requires 10,000 steps. If the number of steps is quadratic in the input data volume, then the complexity is quadratic.
cubic	$O(n^3)$	An operation requires $N^3$ steps, where $N$ characterizes the amount of input data. For example for a 100 elements approximately 1 000 000 steps are executed.
exponential	$O(2^n)$ , $O(N!)$ , $O(n^k)$ , ...	To perform an operation or computation, a number of steps is required that is exponentially related to the size of the input data. For example, at $N=10$ the exponential function $2^N$ has a value of 1024, at $N=20$ it has a value of 1 048 576, and at $N=100$ the function has a value that is a number of about 30 digits.

# Special techniques for analysis of algorithms

## *Using a barometer*

Consider again the following program fragment:

```
unsigned sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i*i; j++)
        sum++;
```

Above we defined its complexity using properties of sums. We can approach it another way: we choose an appropriate instruction (barometer) and watch how many times it is executed.

This vacates us of the concern of analyzing all other instructions that are not relevant to the chosen one. How to choose the barometer?

# Special techniques for analysis of algorithms

## *Using a barometer*

Consider again the following program fragment:

```
unsigned sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i*i; j++)
        sum++;
```

This should be an instruction that is executed at least as often as any other instruction in the program.

In the above program fragment, a suitable candidate for this is `sum++`. By the way, the value of the `sum` variable after the fragment is executed will give us the number of executions of the `sum++` instruction.

# Special techniques for analysis of algorithms

- When analyzing computer algorithms, we most often study their behavior in the **worst** or **average case** and almost never care how they behave in the **best case**.
- A commonly used technique in analyzing an algorithm to determine its worst-case complexity is to assume that the worst possible outcome occurs at each step.
- This gives us a correct complexity of  $O(\dots)$ , but does not always give us a correct estimation: the results are often quite pessimistic, and in practice the algorithm runs much faster even in the worst case.
- The reason is that the worst case may not always occur at *each* step. Thus, the overall complexity may turn out to be significantly better than predicted.

# Special techniques for analysis of algorithms

- Although they give us the opportunity to put the complexity estimation of algorithms and programs on a sound theoretical footing, we should be somewhat suspicious of asymptotic estimates.
- Their main characteristic is that they are interested in the behavior of the algorithm under *infinite* growth of  $n$ . However, there are no *infinitely large numbers* in the real *computing* world.
- This means that the estimate given by the asymptotic function may be inadequate: for example, because it hides the constants, or because we are not interested in such large values of  $n$ .



# Questions and exercises:

1. What is complexity of algorithms?
2. What is time complexity?
3. What is space complexity?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

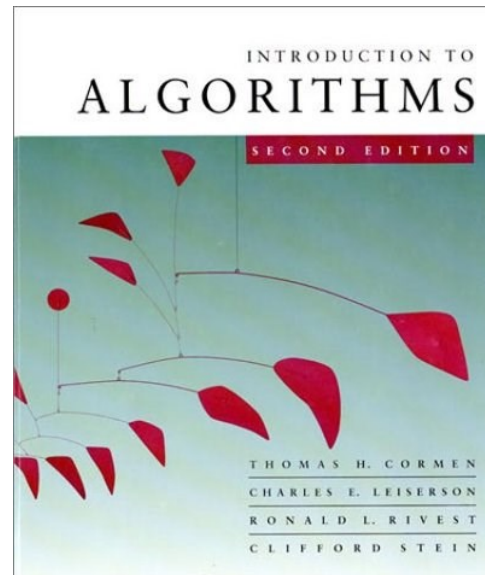
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 2. Strategies in Algorithm Design**
  - ❑ Lesson 1. Divide and Conquer Paradigm in Algorithms



ERASMUS+

# DIVIDE AND CONQUER PARADIGM IN ALGORITHMS



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009).  
Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.

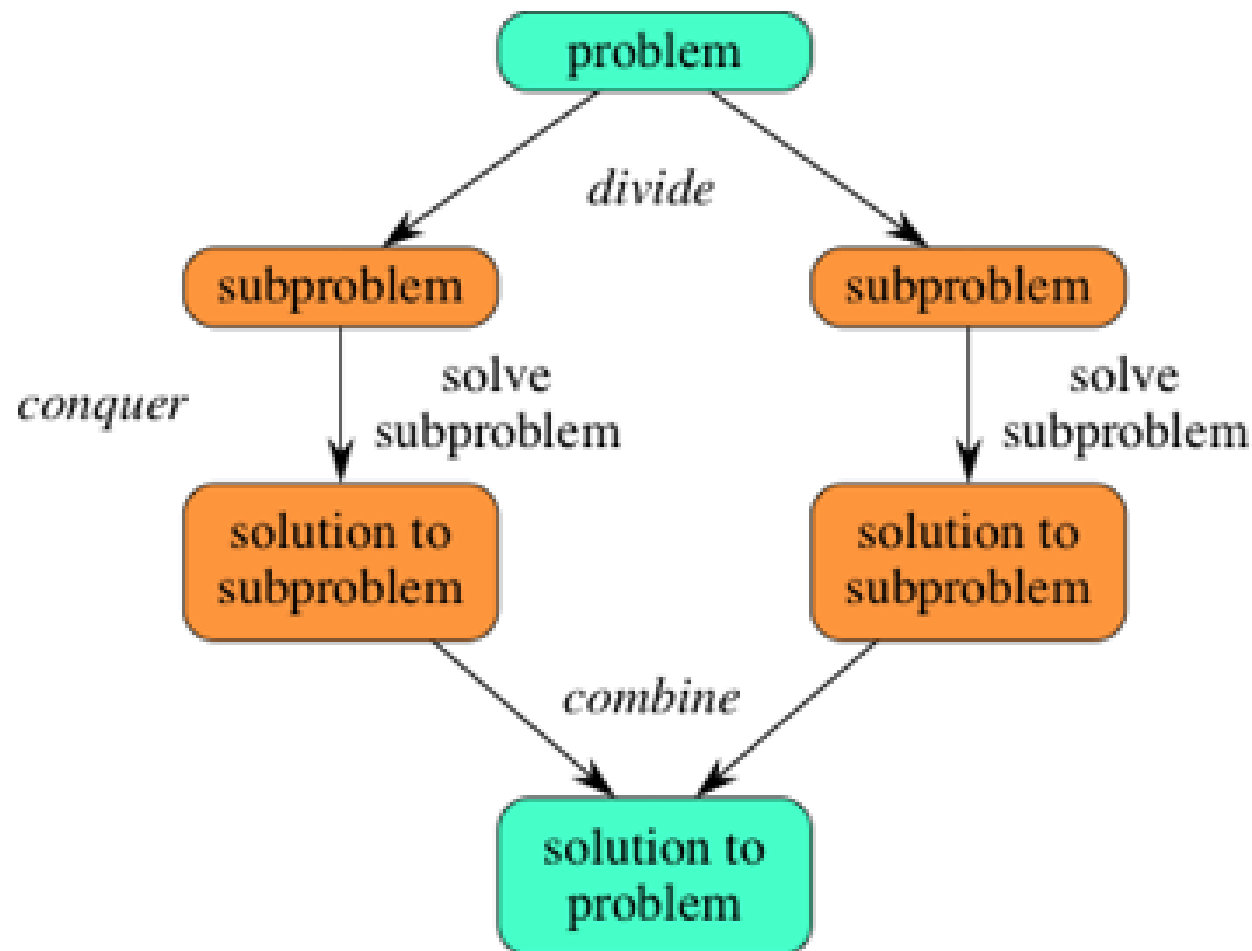
# History

- The roots of the idea of dividing a complex task into several simpler ones, which are easier to attack separately and whose solution allows easy construction of a solution to the initial task, lie far back in antiquity.
- It reached its peak under the Roman Empire, which formulated and promoted “Divide and conquer” as the basic principle of its foreign policy towards the Empire's neighbouring warring tribes.
- The idea of the divide-and-conquer algorithm is to sequentially (**usually by recursion**) break a problem into two or more subproblems until they **become small** enough to be **easily solved**.

# Paradigm of design divide and conquer

1. Division of the problem (task) into subproblems (subtasks).
2. Solve the small subproblems of the problem recursively.
3. Obtain a solution at the original (larger instance) by combining these solutions at the smaller instances.

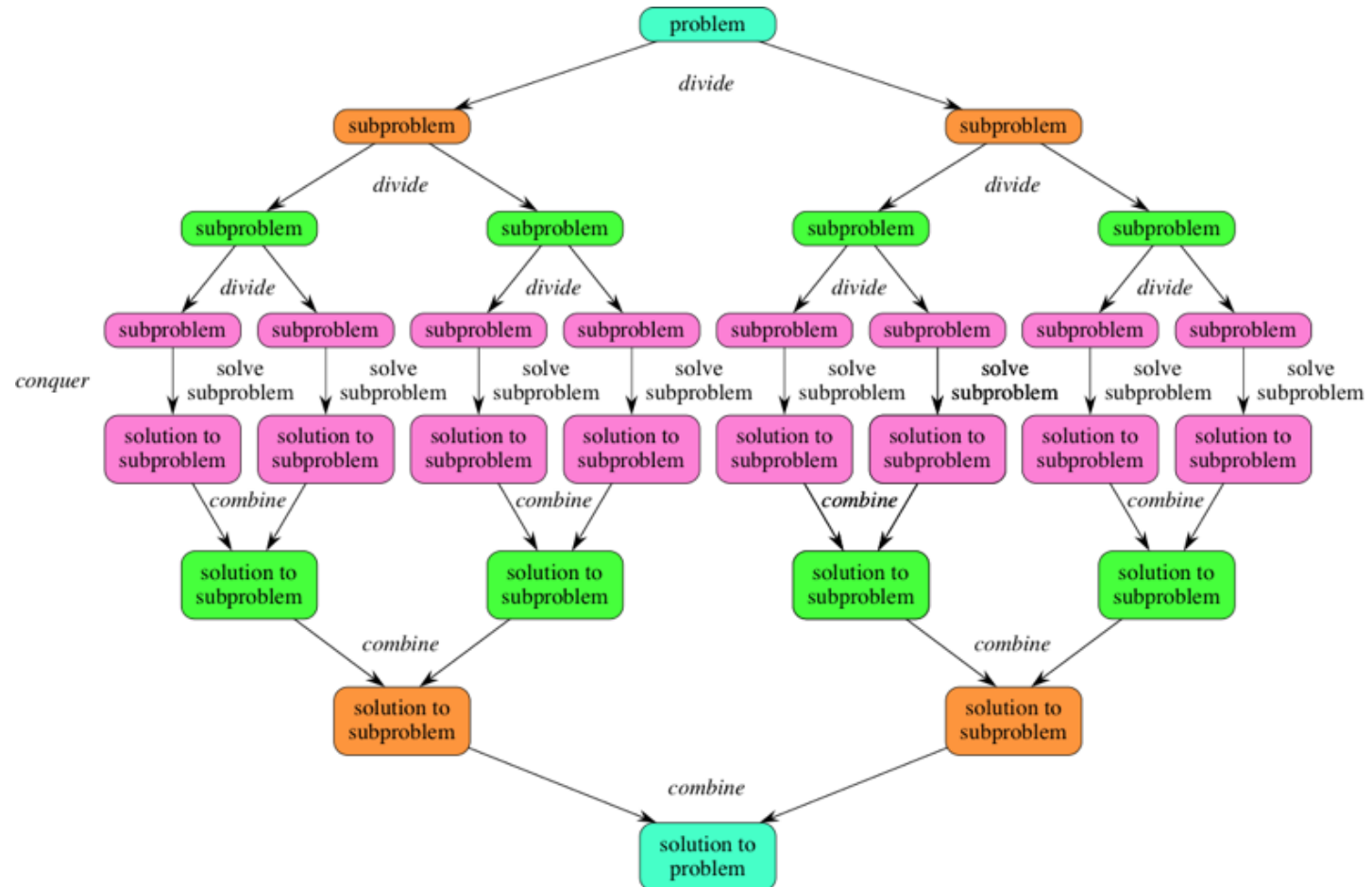
## Divide and Conquer algorithm



ERASMUS+



- **Divide and Conquer algorithm**



ERASMUS+

# Examples of divide and conquer

Typical examples of divide and conquer algorithms are:

- **The sorting algorithms: merge sort and quick sort;**
- **Binary search tree;**
- **Raising a number to a power;**
- **Fibonacci numbers;**
- **Matrix multiplication: the Strassen algorithm.**
- **Divide-and-conquer method can be used for Big Data Analysis.**

# Divide and conquer

## Merge sort:

1. Divide: trivial
2. Conquer: recursive sorting of the two subarrays
3. Combine: merge the two sorted subarrays in linear time.

# Divide and conquer

## Merge sort:

*Step by step example*

6 5 3 1 8 7 2 4

An example of how merge sort sorts a row.

# Divide and conquer

## Merge sort – Basic theorem:

$$T(n) = a T(n/b) + f(n)$$

**CASE 1:**  $f(n) = O(n^{\log_b a - \varepsilon})$ , constant  $\varepsilon > 0$   
 $\Rightarrow T(n) = \Theta(n^{\log_b a})$ .

**CASE 2:**  $f(n) = \Theta(n^{\log_b a} \lg^k n)$ , constant  $k \geq 0$   
 $\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .

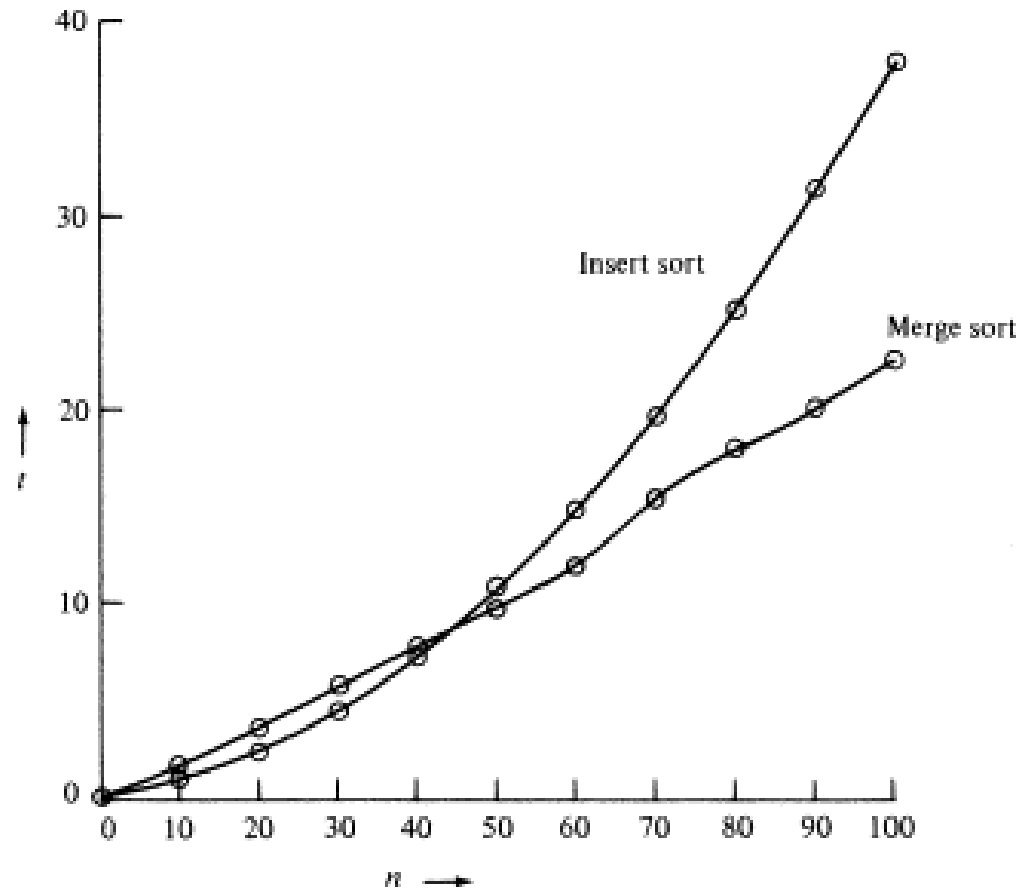
**CASE 3:**  $f(n) = \Omega(n^{\log_b a + \varepsilon})$ , constant  $\varepsilon > 0$ ,  
and regularity condition  
 $\Rightarrow T(n) = \Theta(f(n))$ .

**Merge sort:**  $a = 2, b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$   
 $\Rightarrow$  **CASE 2** ( $k = 0$ )  $\Rightarrow T(n) = \Theta(n \lg n)$ .

ERASMUS+

# Divide and conquer

## Merge sort



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# Divide and conquer

## Binary search:

Search for an element in a sorted array:

1. Divide: **checking the middle element**
2. Conquer: **recursive search in 1 subarray**
3. Combine: trivial



# Divide and conquer

## Binary search:

*Example: searching for 9:*

3	5	7	8	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	12	15

# Divide and conquer

Binary search:

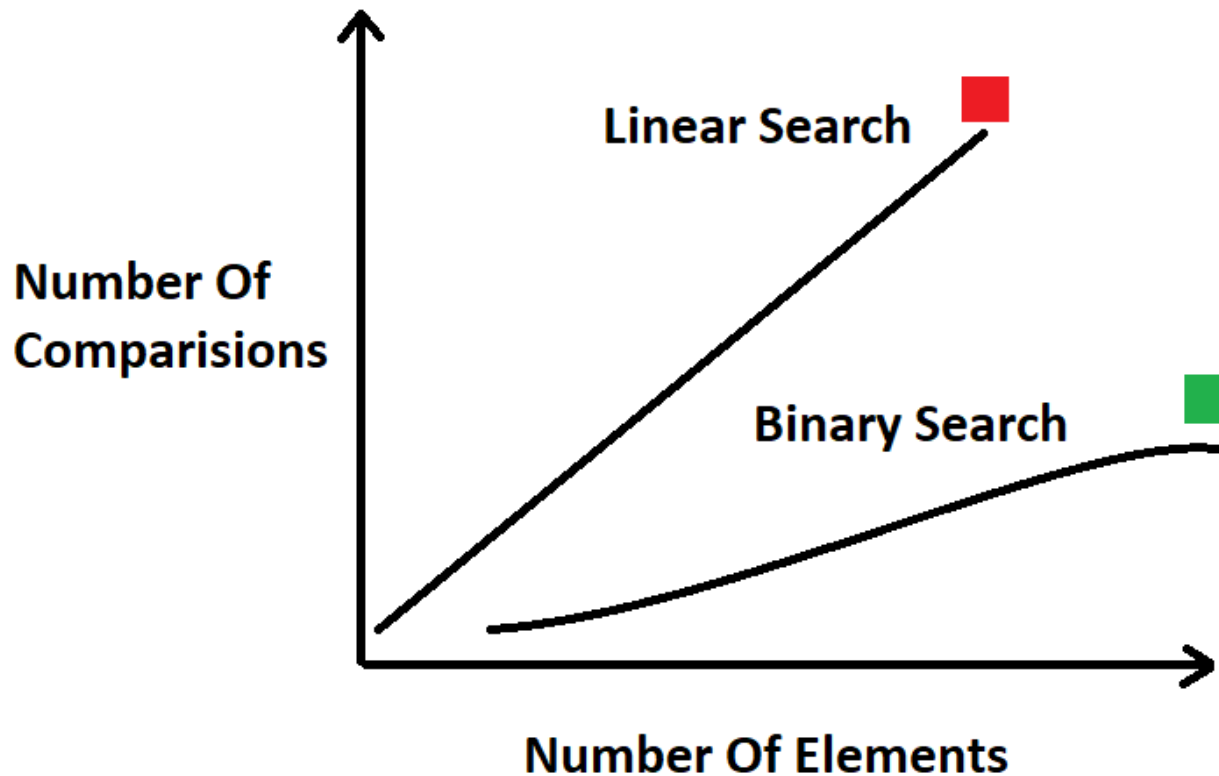
$$T(n) = 1T(n/2) + \Theta(1)$$

*# subproblems*      *subproblem size*      *work dividing and combining*

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0) \\ \Rightarrow T(n) = \Theta(\lg n) .$$

# Divide and conquer

Binary search:



# Divide and conquer

**Raising a number to a power:**

**Проблем:** calculating  $a^n$ , where  $n \in \mathbb{N}$ .

**Naïve algorithm:**  $\Theta(n)$ .

temp=power (base, expo/2) ?

**Divide and conquer algorithm:**

Approach	Time Complexity	Space Complexity
Naive Iterative Approach	$O(n)$	$O(1)$
Divide and Conquer	$O(n)$	$O(\log n)$
Optimized Divide and Conquer	$O(\log n)$	$O(\log n)$

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is an even number} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is an odd number} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n)$$

# Divide and conquer

Fibonacci numbers:

**Recursive definition:**

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

0 1 1 2 3 5 8 13 21 34 ...

---

**Algorithm 1:** F() Recursive

---

**Data:**  $N$

**Result:**  $F(N)$

**if**  $N == 0$  **or**  $N == 1$  **then**  
| **return**  $N$

**end**

**return**  $F(N-1) + F(N-2)$

---

**Naive recursive algorithm:**

(exponential time), where  
is the golden ratio.

$$\Omega(\phi^n)$$

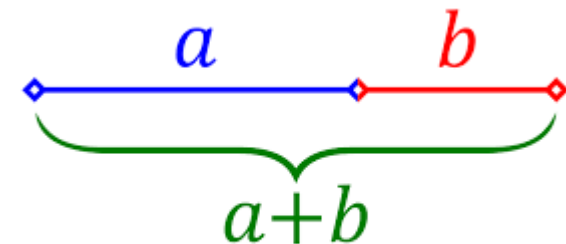
$$\phi = (1 + \sqrt{5}) / 2$$

# Divide and conquer

The Golden Ratio (also known as the Golden proportion, Golden coefficient, or Divine proportion) is an the ratio of the length of the longer part, say "a" to the length of the shorter part, say "b" is equal to the ratio of their sum " (a + b)" to the longer length. It is denoted by the Greek letter  $\phi$  and has a value approximately equal to 1.618...

The ratio of each Fibonacci number to the previous one tends to 1.61803. . . . The number 1.61803. . . is a "golden ratio". It is denoted by the capital Greek letter  $\phi$  (phi).

*The golden ratio search is a divide and conquer technique successfully applied to search for an element in a sorted array.*



$a+b$  is to  $a$  as  $a$  is to  $b$

$$\frac{a+b}{a} = \frac{a}{b} = 1.618... = \phi$$

# Divide and conquer

## Calculation of Fibonacci numbers:

### From the bottom upwards:

We calculate  $F_0, F_1, F_2, \dots, F_n$  in a line forming each value by summing the previous two values.

**Duration:**  $\Theta(n)$ .

---

**Algorithm 4:**  $F()$  Bottom-Up

---

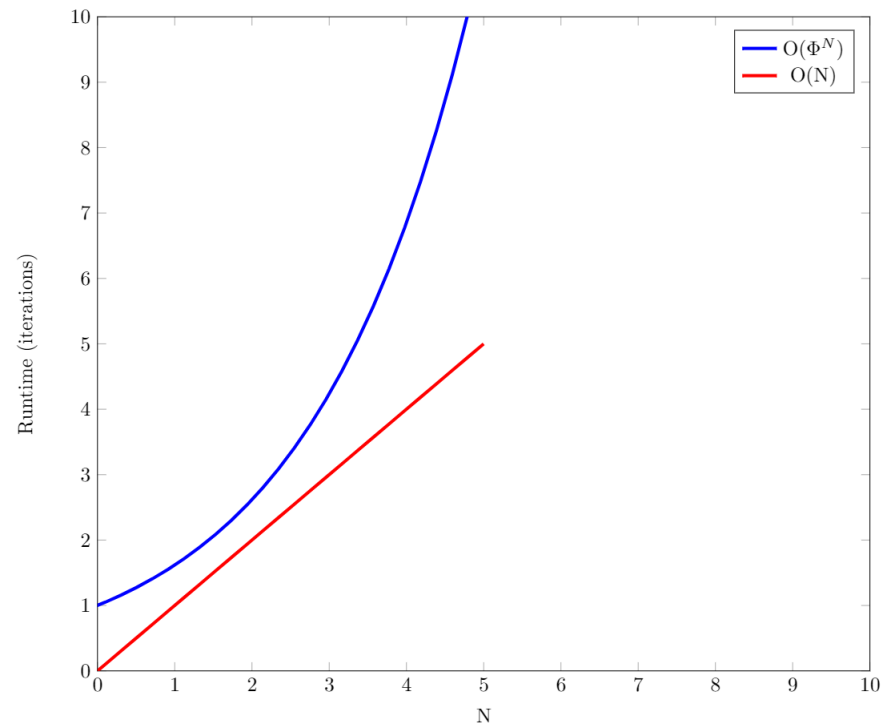
```
Data:  $N$ 
Result:  $F(N)$ 
if  $N == 0$  or  $N == 1$  then
    | return  $N$ 
end
 $A \leftarrow 0$ 
 $B \leftarrow 1$ 
for  $I \leftarrow 2$  to  $N$  do
    |  $Temp \leftarrow A + B$ 
    |  $A \leftarrow B$ 
    |  $B \leftarrow Temp$ 
end
return  $B$ 
```

---



# Divide and conquer

## Calculation of Fibonacci numbers:



# Divide and conquer

## Matrix multiplication:

**Input:**  $A = [a_{ij}], B = [b_{ij}].$   $\left. \vphantom{\begin{matrix} A \\ B \end{matrix}} \right\} i, j = 1, 2, \dots, n.$   
**Output:**  $C = [c_{ij}] = A \cdot B.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

# Divide and conquer

## Matrix multiplication:

### Standard algorithm

*Improving the efficiency of algorithms, especially those in bioinformatics, can have a widespread impact. Matrix multiplication is one such problem, occurring in bioinformatics often.*

```
for  $i \leftarrow 1$  to  $n$ 
  do for  $j \leftarrow 1$  to  $n$ 
    do  $c_{ij} \leftarrow 0$ 
      for  $k \leftarrow 1$  to  $n$ 
        do  $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$ 
```

Running time =  $\Theta(n^3)$

# Divide and conquer

## Matrix multiplication:

## Divide and conquer algorithm

**Idea:**  $n \times n$  matrix =  $2 \times 2$  matrix of  $(n/2) \times (n/2)$  submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\left. \begin{array}{l} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{array} \right\} \begin{array}{l} 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$

# Divide and conquer

**Matrix multiplication:**

**Divide and conquer algorithm**

**Idea:**  $n \times n$  matrix =  $2 \times 2$  matrix of  $(n/2) \times (n/2)$  submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\left. \begin{array}{l} r = ae + bg \\ s = af + bh \\ t = ce + dh \\ u = cf + dg \end{array} \right\} \begin{array}{l} \text{recursive} \\ 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$

# Divide and conquer

**Matrix multiplication:**

**Divide and conquer algorithm**

$$T(n) = 8T(n/2) + \Theta(n^2)$$

*# submatrices*      *submatrix size*      *work adding submatrices*

$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^3).$$

**No better than a simple algorithm.**

# Divide and conquer

## Matrix multiplication:

## Divide and conquer algorithm, Strassen Algorithm

- Multiply  $2 \times 2$  matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$s = P_1 + P_2$$

$$t = P_3 + P_4$$

$$u = P_5 + P_1 - P_3 - P_7$$

7 mults, 18 adds/subs.

**Note:** No reliance on  
commutativity of mult!



# Divide and conquer

## Matrix multiplication:

## Divide and conquer algorithm, Strassen Algorithm

- Multiply  $2 \times 2$  matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$= (a + d)(e + h)$$

$$+ d(g - e) - (a + b)h$$

$$+ (b - d)(g + h)$$

$$= ae + ah + de + dh$$

$$+ dg - de - ah - bh$$

$$+ bg + bh - dg - dh$$

$$= ae + bg$$

# Divide and conquer

## Matrix multiplication:

### Strassen algorithm

1. Divide: dividing  $A$  and  $B$  to submatrices  $(n/2) \times (n/2)$   
Part of the multiplication can be replaced by  $+$  and  $-$ .
2. Conquer: performing 7 multiplications of  $(n/2) \times (n/2)$  submatrices.
3. Combine: for  $C$  we use  $+$  and  $-$  of submatrices  $(n/2) \times (n/2)$

$$T(n) = 7 T(n/2) + \Theta(n^2)$$

# Divide and conquer

## Matrix multiplication:

### Strassen algorithm

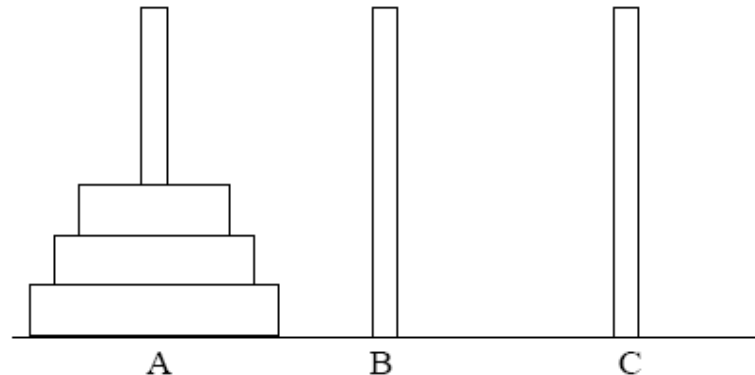
**Strassen's algorithm** is an algorithm used in linear algebra for fast matrix multiplication. For large matrices it is faster than the classic one. It was discovered in the late 1960s by the German mathematician Volker Strassen.

For multiplication of two matrices of size  $N \times N$ , the computation of the resulting matrix is  $\Theta(n^3)$ . Strassen managed to improve it to  $\Theta(n^{\log 7})$ .

# Divide and conquer

## Tower of Hanoi Task

There are  $n$  number of disks of different diameters and three pillars A, B and C. The disks are strung on the first pillar in order of decreasing size and form a tower



They must be transferred from the first pillar to the last, subject to the following rules:

1. On each move, one disc can be moved, and this disc must be the top one for one of the pillars.
2. A disc with a larger diameter cannot be placed on one with a smaller diameter.

# Divide and conquer

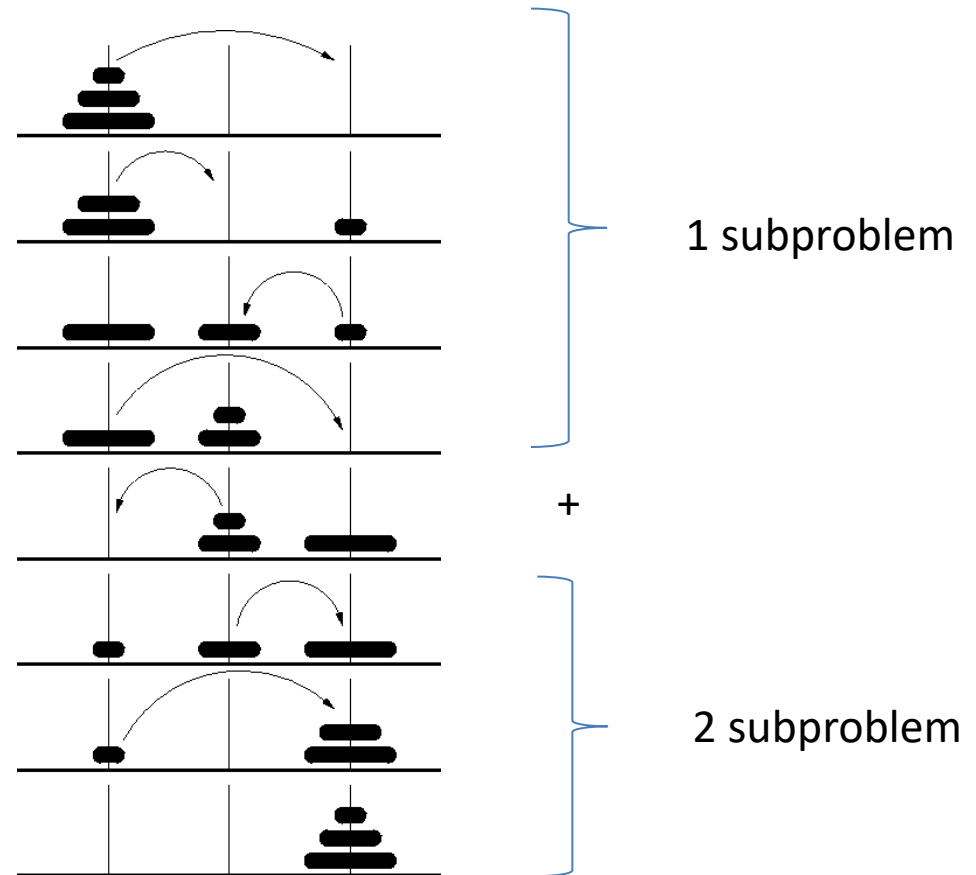
## Tower of Hanoi Task

- The idea is that to move  $n$  number of disks from pillar A to pillar C, we need to move  $n-1$  disks from pillar A to pillar B, then move the disk number  $n$  (which is the largest among them) from pillar A to pillar C and finally move the remaining  $n-1$  disks from pillar B to pillar C.
- Thus, we naturally reduced the task to solving two other instances of it of smaller size (divide), whose union, together with an additional operation (transfer to the largest disk) gives us the solution we are looking for (conquer).

# Divide and conquer

## Tower of Hanoi Task:

- Example with 3 discs



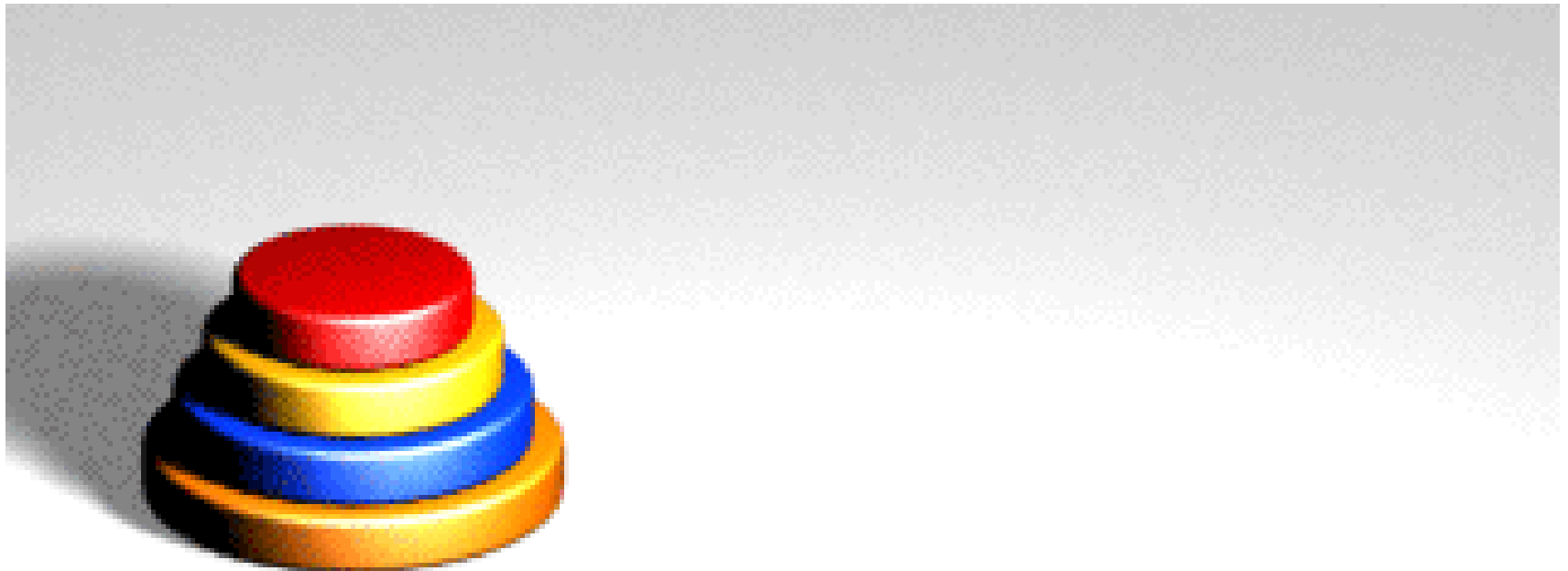
ERASMUS+

# Divide and conquer

## Tower of Hanoi Task:

- Watch how a tower of Hanoi of 4 discs is being solved

*Video demonstration:*





# Divide and conquer

## Conclusion:

- ❑ Divide and conquer is just one of several powerful algorithm design techniques.
- ❑ Divide and conquer algorithms can be analyzed using iterations.
- ❑ The divide-and-conquer strategy often leads to efficient algorithms.

## Questions and exercises:

1. What is the divide and rule strategy ?
2. Can you specify sorting algorithm which using the split and conquer strategy?
3. Can you specify some other popular problems solved by divide and conquer ?



# Thank you for your attention !

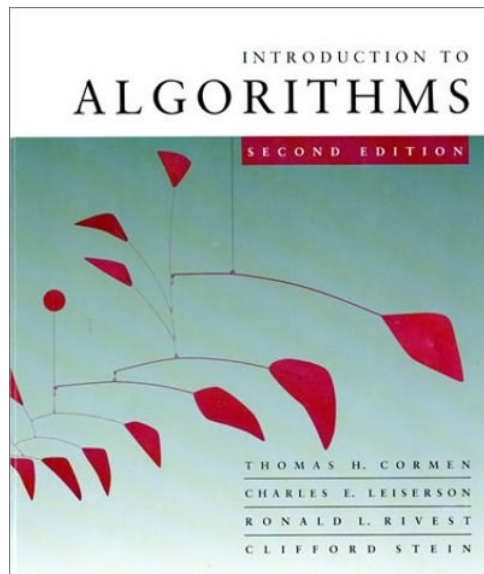
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 2. Strategies in Algorithm Design**
  - ❑ Lesson 2. Dynamic programming



ERASMUS+

# DYNAMIC PROGRAMMING



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT  
Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov  
(2002). Programming = ++Algorithms). Top Team  
Co, София.

# Dynamic Programming

- **Dynamic programming DP** (optimization, DO), like the divide-and-conquer method, solves problems (task) by combining solutions to subproblems. ("Programming" in this context refers to a tabular method, not written computer code).
- As we saw in the “*Divide and Conquer*” lecture, these algorithms divide the problem (task) into unconnected subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem.

# Dynamic Programming

- In contrast, **dynamic programming** is applied when subproblems overlap, that is, when subproblems share sub-subproblems.
- In this context, a divide-and-conquer algorithm, does more work than necessary, repeatedly solves common subproblems.
- A **dynamic programming** algorithm solves each subproblem only once and then records its answer in the form of a **table**, thus avoiding the work of recalculating the answer each time when solving each subproblem.



# Dynamic Programming

- We typically use dynamic programming for optimization problems. Such problems may have many possible solutions. Every solution has a value, and we want to find the solution with the optimal (minimum or maximum) value.
- We call such a solution an "**optimal solution to the problem**" rather than an "optimal solution", because there may be several solutions for which an **optimal value** can be achieved.

# Dynamic Programming

- When developing a dynamic programming algorithm, we follow a four-step sequence:
  1. Characterize the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of an optimal solution, typically in a bottom-up model.
  4. Construct an optimal solution from computed information.

# Dynamic Programming

- Steps 1-3 are the basis of dynamic programming to solve a problem.
- If we are only interested in the value of an optimal solution and not the solution itself, then we can skip step 4.
- ***Definition.*** *The programming technique in which a table is filled up with the results of solutions to subproblems already solved to avoid repeated computations is called dynamic programming.*

# Dynamic Programming

- Dynamic programming is based on solving subproblems of the initial problem with a smaller size and **safe the already computed results**, i. e. speed is gained at the expense of memory.
- In some cases a constant memory is needed (Fibonacci numbers), while in other cases the memory needed can be linear (Knapsack problem), quadratic (optimal matrix multiplication problem), and sometimes even larger.

# Dynamic Programming

- It should also be noted that dynamic programming is not always applicable.
- On the one hand, the solution of the initial problem cannot always be obtained by combining the results of solving some or all of its subproblems.
- On the other hand, even when such a combination is possible, the number of subtasks to be considered may be unacceptably large.
- To this it should be added the lack of a clear criterion characterizing the problems that can be solved using the described method.

# Dynamic Programming

- It turns out that for a number of tasks, standard algorithms turn out to be far more efficient than dynamic programming, and for others – this method is not applicable at all.
- There are two necessary conditions for the method application: *optimal solution substructure* and *overlapping subproblems*.
  - *The optimal solution substructure means that the optimal solution of the initial problem can be found as a function of the optimal solutions of the subproblems. This leads to a strong limitation of the set of subtasks, hence to a higher efficiency of the method.*

# Dynamic Programming

- The second necessary condition for the application of dynamic programming is *overlapping the subtasks*. Dynamic programming is able to make clever use of overlapping subproblems, computing the solution to each subproblem only once, thus severely limiting the actual number of subproblems solved.
- The less often a new subproblem actually needs to be solved, the more efficient the method is. The lack of overlap of subtasks is a sure indicator that the application of dynamic programming is inappropriate. In such a case, it is better to try the divide and conquer method.



# Dynamic Programming

- Often the objects considered in solving a problem are sets with partial or complete linear ordering of the elements.
- Examples of such objects are strings, various combinatorial configurations (permutations, combinations, etc. ), leaves of ordered search trees, points on a line, vertices of a polygon, etc.
- In these cases, dynamic programming usually results in an efficient algorithm.

# Dynamic Programming

- Typically, in dynamic programming, problems are solved iteratively from the bottom-up, i. e. , trivial subproblems (those that are solved directly without further decomposition) are considered first, then those larger subproblems for which all subproblems have already been solved, and so on.
- The main disadvantage of this approach is that it requires solving all subproblems of a size smaller than that of the initial problem, which leads to redundant computations, because in the process of computation it may turn out that some of the subproblems are not needed.

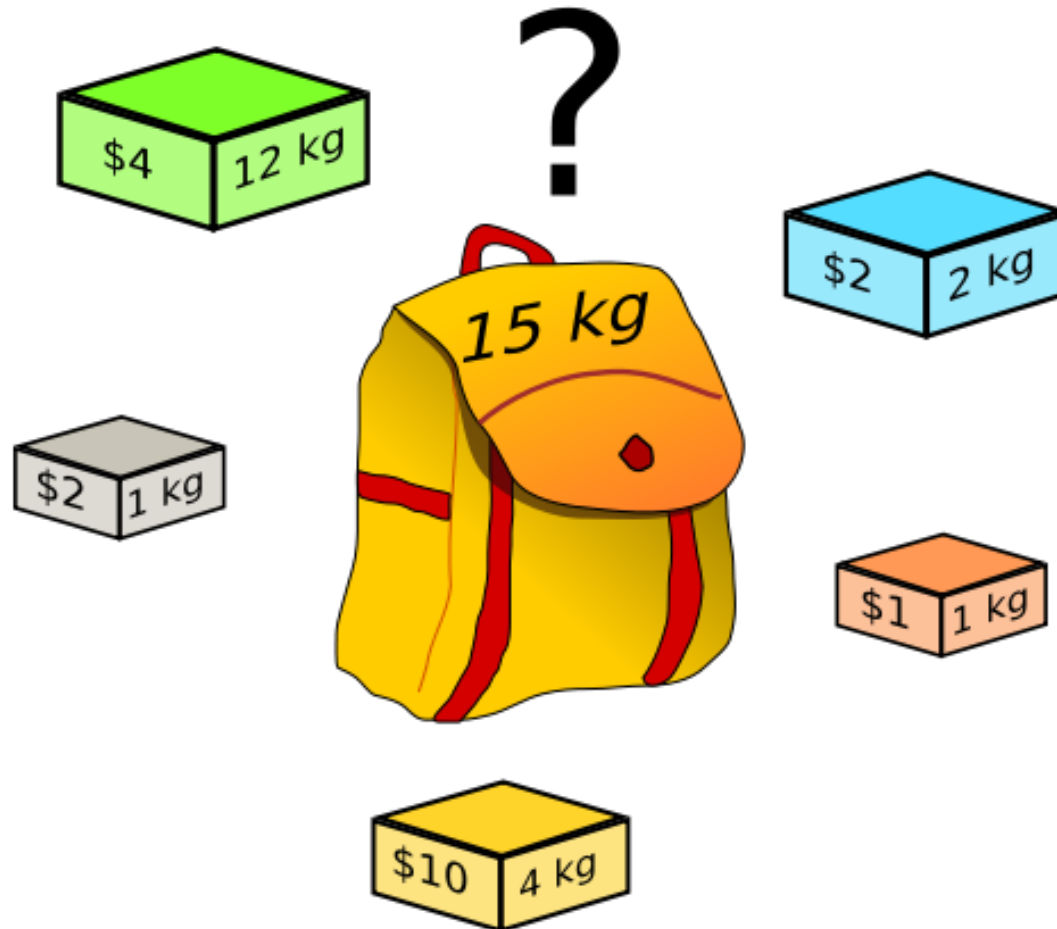
# Dynamic Programming

- Dynamic programming is a good method for **optimizing join queries in relational database management systems**, and virtually all commercial optimizers rely on dynamic programming for this purpose.
- Dynamic programming is often underestimated, or more accurately - undervalued. Some programmers consider it complicated and messy.
- In fact, it is one of the simplest yet effective algorithmic techniques.
- Since such knowledge is most easily learned on the basis of concrete examples, we will consider a classic example.

# Knapsack problem

- This is one of the most famous problems in the world of dynamic programming, solved efficiently using the method.
- The knapsack problem appears in the literature in many different formulations, suggesting different solutions.
- In its most popular version (the 0-1 knapsack problem), it is a classic example of a problem that is solved with **dynamic programming**.

# Knapsack problem



ERASMUS+

# Knapsack problem

Given a knapsack with a capacity of  $M$  kilograms and  $N$  items for each of which are given two weights  $m_i$  and a value  $c_i$ . Choose set of items whose total value is maximum and the sum of their weights does not exceed  $M$ .  $M$ ,  $N$ ,  $c_i$  and  $m_i$  are natural numbers ( $1 \leq i \leq N$ ).

The task comes down to finding the maximum of the sum:

$$\sum_{i=1}^N x_i c_i$$

under restricted conditions:

$$\sum_{i=1}^N x_i m_i \leq M,$$

$$c_i > 0, m_i > 0, x_i \in \{0,1\}, i = 1, 2, \dots, n$$

Restrictions on  $c_i$  and  $m_i$  are restrictions on condition of the task (*on the concrete example*) while these on  $x_i$  should be taken as restrictions on the *solution sought*.

Solution: We will define a recurrent target function  $F(i)$ , that is a solution for a knapsack with a capacity  $i$ .

Then:

$$F(i) = \begin{cases} 0 & i = 0 \\ \max_{j=1,2,\dots,N; m_j \leq i} [c_j + F(i - m_j)] & i > 0 \end{cases}$$

ERASMUS+

# Knapsack problem

Judging from the above recurrent formula, we can implement a function to calculate  $F$ . We will point out that this way we can choose one and the same item multiple times. That's why for each  $F_{(i)}$  we will maintain set  $[i]$ , containing the concrete items which taking leads to this maximum value. Except let us find a concrete set of items that forms the maximum value of the target function set  $[i]$  will prevent us from including one and same item again..

A possible approach is to calculate the function value recursively and in order to avoid repeated calculations to use a table (memorization). The table of already calculated values will contain the value of the target function if it has already been calculated or a special value NOT\_CALCULATED – otherwise. Every time we need the value of  $F(i)$  for some  $i = 1, 2, \dots, N$ , we will first check whether  $Fn[i]$  (saved table value) is different than NOT\_CALCULATED in which case we're taking the value  $Fn[i]$  from the table. We will set the program input data in its code as constants. The main job is done by the recursive function  $F()$  which calculates the values of the target function.

It is done by the function calculate () which checks whether the capacity of the knapsack enables taking all item, and only if it is not like this, turns to  $F()$ .



# Knapsack problem

Example with 7 items and knapsack with a capacity 19.

capacity $\Rightarrow$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
no items	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<u>value=90 weight=9</u>	0	0	0	0	0	0	0	0	0	90	90	90	90	90	90	90	90	90	90	90
value=85 weight=10	0	0	0	0	0	0	0	0	0	90	90	90	90	90	90	90	90	90	90	175
value=85 weight=10	0	0	0	0	0	0	0	0	0	90	90	90	90	90	90	90	90	90	90	175
value=70 weight=15	0	0	0	0	0	0	0	0	0	90	90	90	90	90	90	90	90	90	90	175
<u>value=89 weight=9</u>	0	0	0	0	0	0	0	0	0	90	90	90	90	90	90	90	90	90	179	179
value=89 weight=12	0	0	0	0	0	0	0	0	0	90	90	90	90	90	90	90	90	90	179	179
value=100 weight=9	0	0	0	0	0	0	0	0	0	100	100	100	100	100	100	100	100	100	190	<u>190</u>

# Knapsack problem

<http://karaffeltut.com/NEWKaraffeltutCom/Knapsack/knapsack.html>

**Online 0/1 Knapsack problem solver**

Using dynamic programming with javascript [Read about it at wikipedia](#) [Youtube explanation movie-film](#)

capacity =>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
no items	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
delete value=15 weight=3	0	0	0	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
delete value=6 weight=8	0	0	0	15	15	15	15	15	15	15	15	21	21	21	21	21	21	21	21
delete value=4 weight=9	0	0	0	15	15	15	15	15	15	15	15	21	21	21	21	21	21	21	21

**Add Item**  
Value: 4  
Weight: 9  
OK

**Delete all Items**  
delete

**Reload demo items**  
reload

**New capacity**  
knapsack size:  
18  
OK

**Reverse Items**  
OK

**Items in solution**  
val=6 w=8  
val=15 w=3  
solution: 21

Online calculator with which you can test different examples:

<https://knapsack.masao.io/>

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

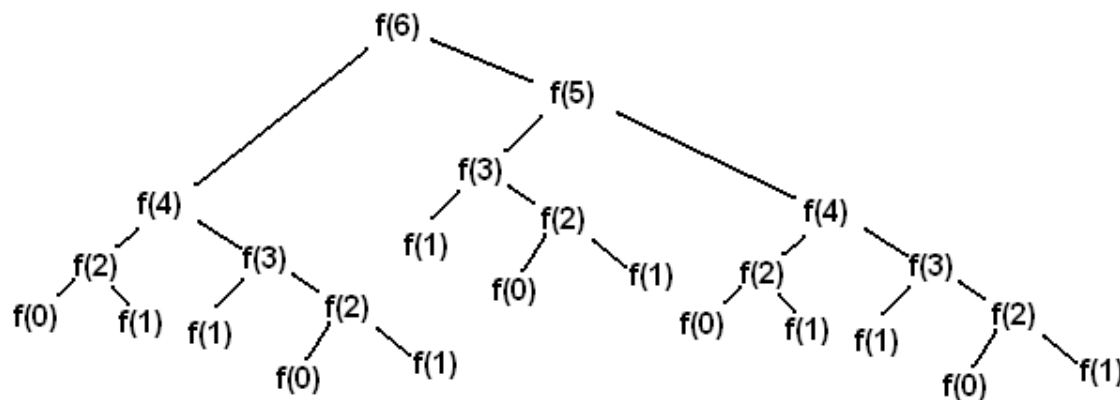
Action Type KA226 - Partnerships for Digital Education Readiness

# Fibonacci numbers

- Calculation with recursive function:

```
int F(int n)
{if(n==0) return 0;
 else if(n==1) return 1;
 else return F(n-1)+F(n-2);}
```

- Too much calculations:



# Fibonacci numbers

- The computation of the Fibonacci numbers can be arranged so that linear time by  $n$  is needed, by storing all the computed values in an array  $F[i]$ :

```
F[0]=0;  
F[1]=1;  
for(i=2;i<=n;i++) F[i]=F[i-1]+F[i-2];
```

- This program fragment illustrates the basic idea - how one can compute Fibonacci numbers sequentially from smaller to larger and at the same time keep the previous results so that when we need to compute  $F_n$ , we already have  $F_{n-1}$  and  $F_{n-2}$  computed and can use them.

# Fibonacci numbers

- We can immediately notice that we do not need to save all the numbers calculated up to the current moment to calculate  $F_n$ .
- It is sufficient to have only the two previous values available. This can be done without using an array, but just two variables, and swapping their values appropriately:

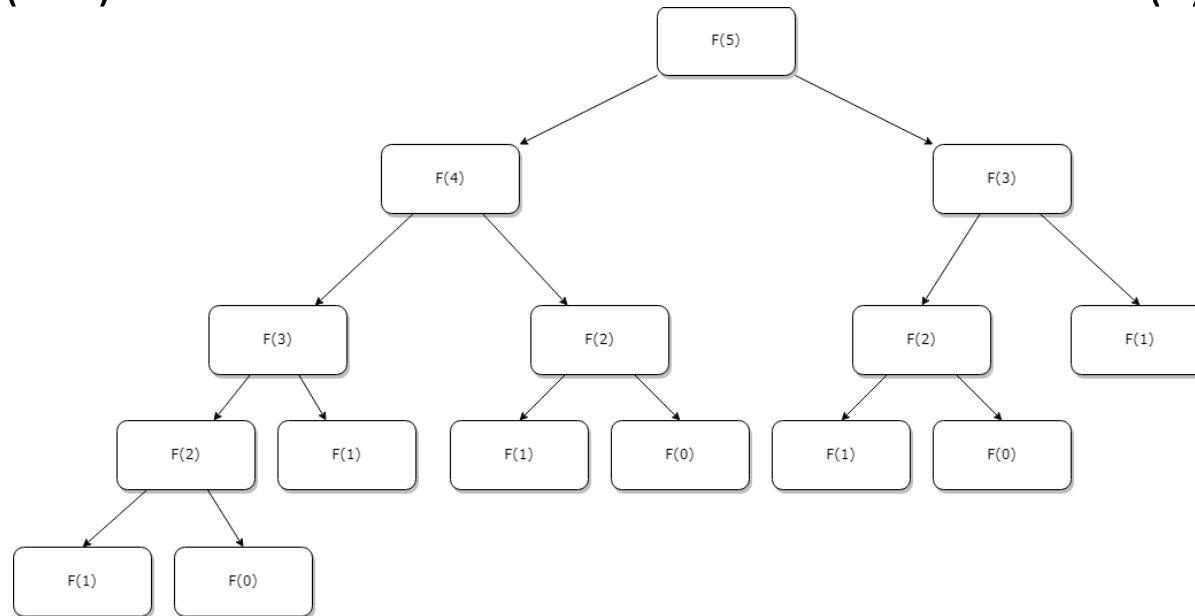
```
f0=0;  
f1=1;  
for(i=2;i<=n;i++) {f=f1+f0; f0=f1; f1=f;}
```



# Fibonacci: Top-Down vs Bottom-Up Dynamic Programming

- **The Recursive Approach:**

- To compute  $F(N)$  in the recursive approach, we first try to find the solutions to  $F(N-1)$  and  $F(N-2)$ . But to find  $F(N-1)$ , we need to find  $F(N-2)$  and  $F(N-3)$ . This continues until we reach the base cases:  $F(1)$  and  $F(0)$ .

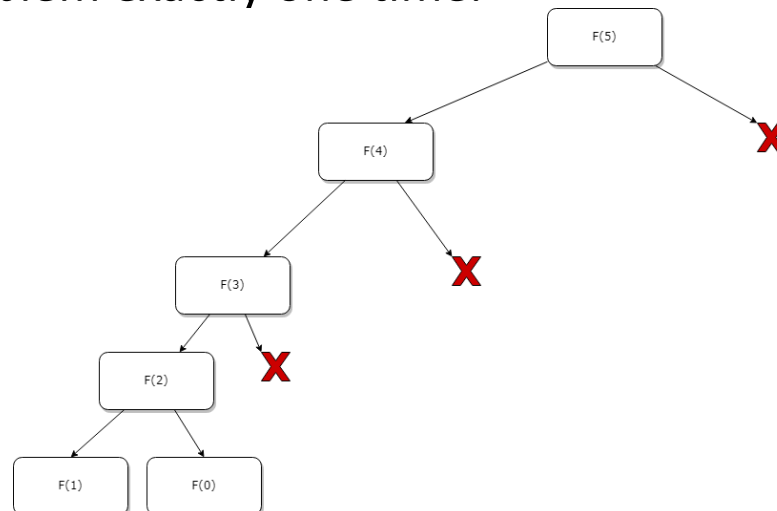




# Fibonacci: Top-Down vs Bottom-Up Dynamic Programming

- **The Top-Down Approach:**

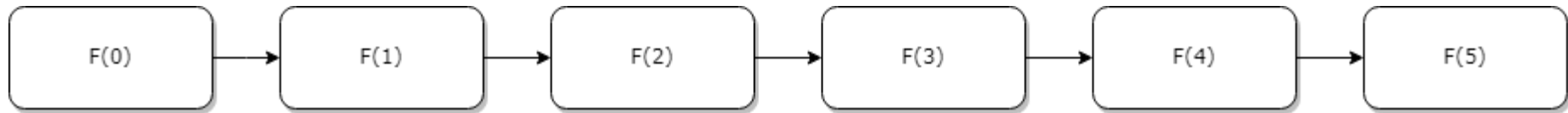
- The idea here is similar to the recursive approach, but the difference is that we'll save the solutions to subproblems we encounter. This way, if we run into the same subproblem more than once, we can use our saved solution instead of having to recalculate it. This allows us to compute each subproblem exactly one time.



# Fibonacci: Top-Down vs Bottom-Up Dynamic Programming

- **The Bottom-Up Approach:**

- In the bottom-up dynamic programming approach, we'll reorganize the order in which we solve the subproblems. We'll compute  $F(0)$ , then  $F(1)$ , then  $F(2)$ , and so on:



- This will allow us to compute the solution to each problem only once, and we'll only need to save two intermediate results at a time.
- For example, when we're trying to find  $F(2)$ , we only need to have the solutions to  $F(1)$  and  $F(0)$  available. Similarly, for  $F(3)$ , we only need to have the solutions to  $F(2)$  and  $F(1)$ .

# Questions and exercises:

1. What is the DO strategy ?
2. What is memorization approach in DP ?
3. Can you specify some other popular problems solved by DO?



# Thank you for your attention!

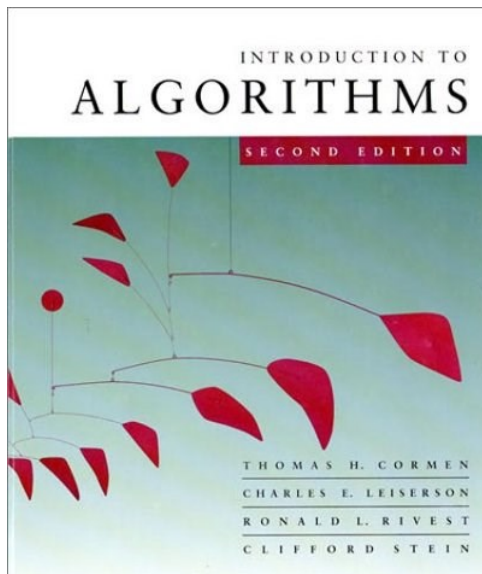
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 2. Strategies in Algorithm Design**
  - ❑ Lesson 3. Heuristic and probability algorithms



ERASMUS+

# HEURISTIC AND PROBABILISTIC (RANDOMIZED) ALGORITHMS



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT  
Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov (2002).  
Programming = ++Algorithms). Top Team Co, Sofia.

# Heuristic and probabilistic algorithms

- Heuristics (from **Ancient Greek**) - find, discover, refers to methods of solving problems, learning or drawing conclusions **based on experience**.



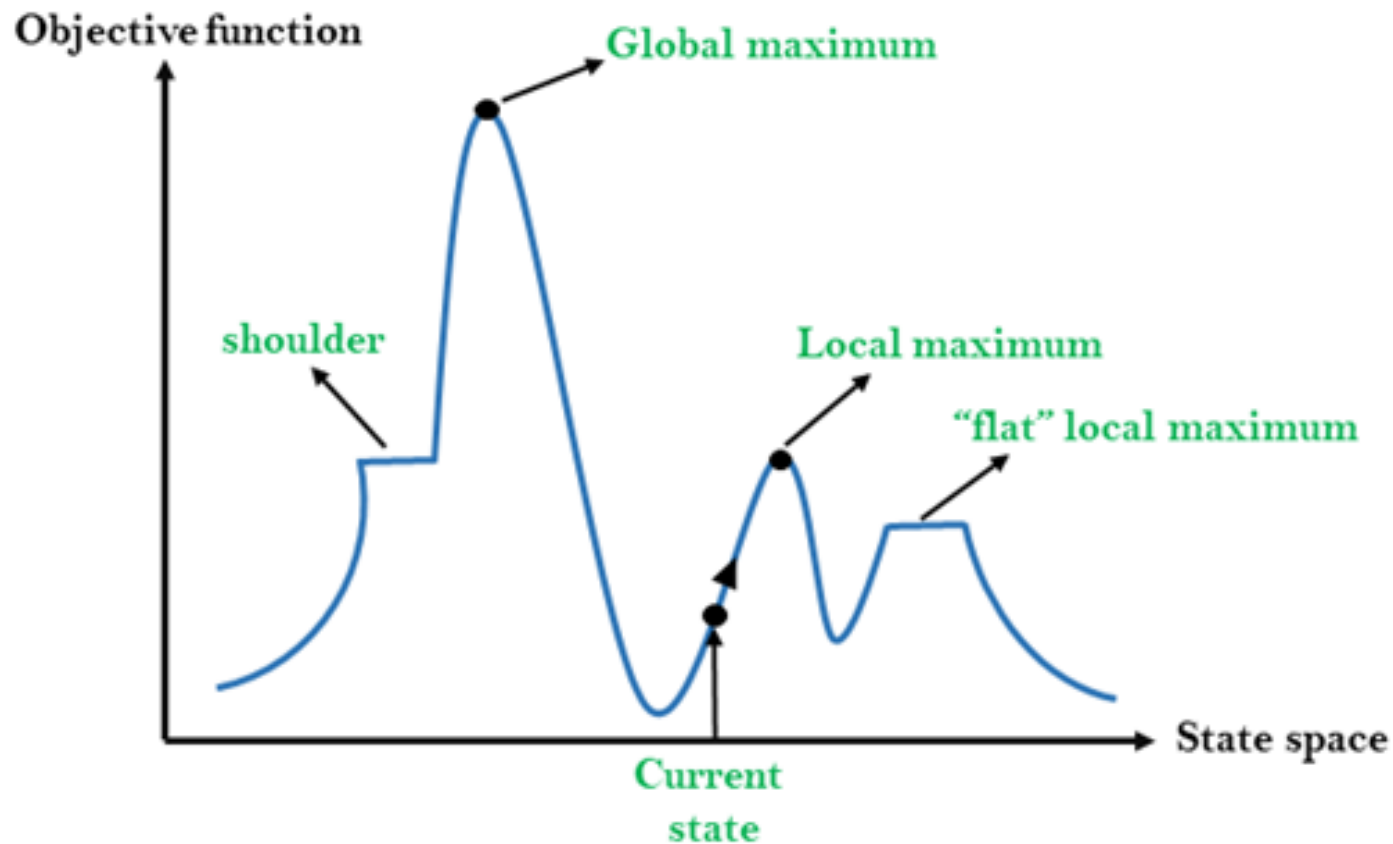
- These methods are not guaranteed to be optimal, they are just a way to reach a conclusion in a faster way and to **reduce complexity**.
- **Heuristic methods are used to speed up the process of finding a good solution where a detailed study is **impractical**.**
- Examples of this method include the use of assumption, based on knowledge, intuitive reasoning.



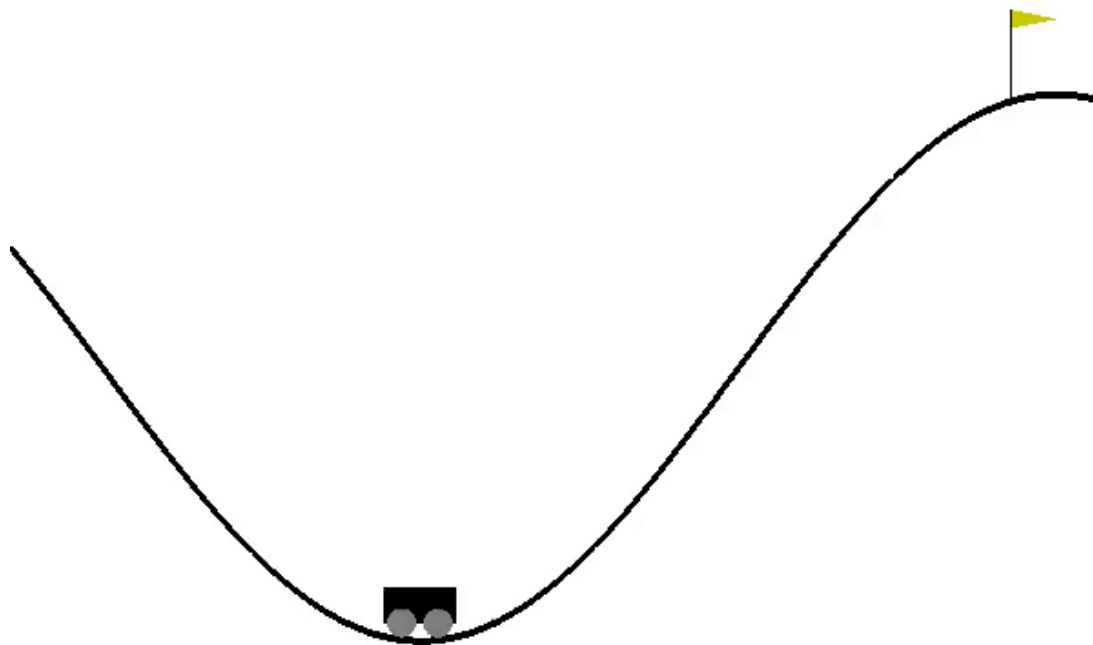
# Heuristic and probabilistic algorithms

- In **computer science**, a heuristic or heuristic method is a method for solving logical or mathematical problems for which there is no **algorithm**. The method involves a stepwise narrowing of the solution-finding domain through **inductive** reasoning based on experience.
- **Hill climbing** is a type of local search algorithm. The basic idea is to find a point in a given space that is better than all others for a given evaluation function. This is also the key heuristic used. The initial state is chosen **randomly** as a point in the search space. All its **neighboring states** are generated. On the basis of the evaluation function, the point for which the function value is the **largest** (or **smallest**, depending on the problem) is selected.

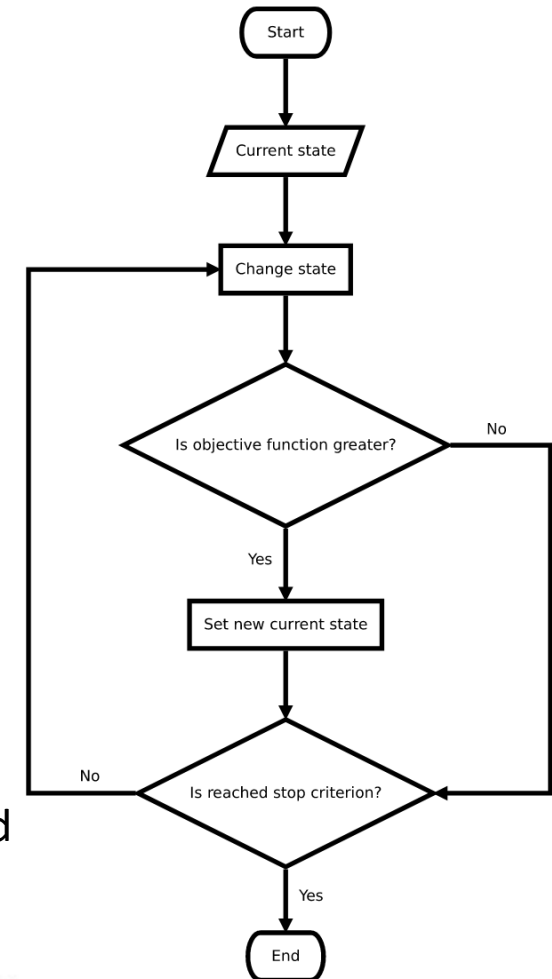
# Heuristic and probabilistic algorithms



# Heuristic and probabilistic algorithms



The algorithm ends when we reach the predefined conditions or when we cannot find further improvements



# Heuristic and probabilistic algorithms

*A **probabilistic algorithm** is one that makes **random choices during its execution**. The operation of such an algorithm can be random, even with fixed input.*

- To find an approximate solution, one random initial solution is generated, then successive iterations produce more accurate solutions than the previous ones.
- The design and analysis of probabilistic algorithms are focused on working independently of what we have as input, and depend on the arbitrary choices made by the algorithm during its run.

***Heuristic (approximate)** algorithms are algorithms whose main task is to find **not an optimal** but **an approximate** solution under memory or time constraints. Examples of such algorithms are local search, tabu search, a class of heuristic algorithms called **greedy** algorithms, etc.*

# Heuristic and probabilistic algorithms

- Suppose you need to hire a new office assistant. Your previous attempts to recruit haven't been successful, and you have decided to use an employment agency.
- The employment agency sends you one candidate every day.
- You can conduct an interview and then decide whether to hire that person or not. You must pay the employment agency a *small fee* to interview the candidate.
- However, you must dismiss your current assistant and you must pay a substantial fee to the employment agency.
- You commit **to have the best office assistant possible** at all times.

# Heuristic and probabilistic algorithms

- Therefore, you should decide that after interviewing each candidate, if that candidate is better qualified than the current office assistant, to dismiss the current office assistant and hire the new assistant.
- You are willing to pay the necessary price for this strategy, but you want to evaluate how much it will cost you.

*Pseudo-code of the hiring procedure:*

```
HIRE-ASSISTANT( $n$ )  
1   $best = 0$            // candidate 0 is a least-qualified dummy candidate  
2  for  $i = 1$  to  $n$   
3      interview candidate  $i$   
4      if candidate  $i$  is better than candidate  $best$   
5           $best = i$   
6      hire candidate  $i$ 
```

# Heuristic and probabilistic algorithms

Pseudo-code of the hiring procedure with random changing the positions of the elements in the array:

(In this case we will have a random permutation of the list of candidates)

RANDOMIZED-HIRE-ASSISTANT( $n$ )

```
1  randomly permute the list of candidates
2   $best = 0$            // candidate 0 is a least-qualified dummy candidate
3  for  $i = 1$  to  $n$ 
4      interview candidate  $i$ 
5      if candidate  $i$  is better than candidate  $best$ 
6           $best = i$ 
7      hire candidate  $i$ 
```



# Greedy algorithms

- In most of the problems, the **optimal** among the possible solutions is sought.
- In many cases, to find the optimal solution, it is necessary to find the solutions of all **subcases of the problem** (not necessarily optimal).
- A major disadvantage is that some of them are possibly **recalculated multiple times**.
- **Repeatedly** computing the same subcases can be avoided by applying **dynamic optimization**, but unfortunately the latter involves the need for sufficient memory to store the results.

# Greedy algorithms

- The **idea** behind heuristic algorithms is the following: the heuristic algorithm targets one of all subcases of the problem and solves only that one, in the "hope" that it will turn out to be the correct one.
- The selection of this sub-case is based on a **local** optimality criterion.
- For example, **greedy algorithms**, as the name suggests, always focus on the **best choice for the moment**, looking locally, and quite naturally, at a later stage, it may turn out that this choice was not the best one, looking globally.

# Greedy algorithms

- Greedy algorithms are easy to compose, the corresponding implementation of the algorithm is not complicated, and the only **disadvantage** is that sometimes they do not guarantee the correct solution of the problem.
- However, the above does not diminish their usefulness - it is characteristic of heuristic algorithms (and greedy ones in particular) that they **quickly manage to find a solution close to optimal.**
- In many practical tasks, it is impossible to explore all cases, and often formulating an algorithm that finds a solution 5% worse than the optimal solution is considered a success, compared to the alternative of an almost infinite and unpromising search for the "true" optimal solution.

ERASMUS+

# Greedy algorithms

- **The first problem** we will consider is the following: find a way to obtain a given sum  $m$  ( $m$  is a natural number), using a minimum number of banknotes, with denominations from the set  $C = \{a_1, a_2, \dots, a_n\}$ . For example, the banknotes are 1, 2, 5, 10, 20, 50 leva.
- *Consider the following algorithm:*
  - 1) We initialize  $s=0$
  - 2) We find the banknote  $i$  with a maximum value  $a_i$  ( $a_i \in C$ ), so that  $s+a_i \leq m$ .
    - 2.1) If there isn't a banknote for which  $s+a_i \leq m$  it follows that the problem has no solution. End.
    - 2.2) Otherwise we take the banknote  $i$  and increase  $s$  by  $a_i$ .
      - 2.2.1) If  $s=m$  it follows that the problem is solved. End.
      - 2.2.2) If  $s < m$  than we still haven't obtained the whole sum and repeat step 2).
- For example, for the sum of 298 leva, five 50 leva notes, two 20 leva notes, one 5 leva note and one two and one leva note will be chosen in turn (total  $250 + 40 + 5 + 2 + 1 = 298$ ).

# Greedy algorithms

- Obviously, the described algorithm satisfies the criteria of a greedy algorithm: at each step it selects the **maximum** higher-valued banknote, thus aiming to achieve the searched amount as fast as possible.
- In **this** particular example, it leads to an **efficient** solution to the problem.
- For example, consider the case where the possible values for notes are 2, 5, 20 and 30 and we want to get a sum of 40.
- The greedy algorithm first will select a note of 30 (the maximum possible), then select two notes of 5, i. e. 3 notes in total.

# Greedy algorithms

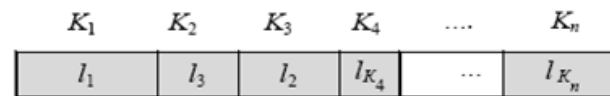
- Obviously, however, there is a better solution: two 20 notes.
- In addition to not finding the optimal solution, it is possible that the greedy algorithm may not find a solution at all.
- So, for example, if we want to get a sum of 6: after the note with value 5, the algorithm is left with no further choices (and the sum can still be obtained from 3 notes with value 2).
- However, things are not always so bad and there are a number of problems where it can be shown that the *greedy algorithm* always finds a solution.

# Greedy algorithms

## Magnetic tape task

There are  $n$  programs of lengths  $l_1, l_2, \dots, l_n$  and a magnetic stripe with sequential access.

- Sequential access means that in order to read a record located at a given position on the tape, you must pass (scroll the tape) to the specified location.



Magnetic stripe with sequential access ( $K_1=1, K_2=3, K_3=2, \dots$ )

- For example, in order to read the third program (of length  $l_2$ ), all programs before it - those of length  $l_1$  and  $l_3$  - must be "scrolled".



# Greedy algorithms

## Magnetic tape task

Given the probabilities of each program  $p_1, p_2, \dots, p_n$ ,  $0 \leq p_i \leq 1$ ,  $1 \leq i \leq n$   $\sum_{i=1}^n p_i = 1$ . For example a probability 0.5 means that the program runs as often as all the other programs together. In fixed ordering  $K_1, K_2, \dots, K_n$  of the programs on the tape the average time needed for loading a random program is defined the following way:

$$T = \frac{\sum_{i=1}^n \left( p_{K_i} \sum_{j=1}^i l_{K_j} \right)}{n}$$

The task is to find an ordering of the programs on the tape which has to minimize  $T$ .

We're going to use a greedy algorithm. It is intuitively clear that as often a program is used as forward on the tape it should be written. The other criterion is the length of the program - as longer it is as backward on the tape it should be (to "hinder" as little as possible when scrolling is needed). Thus, we reach the following algorithm: We write the programs sequentially on the tape and in each step we choose the program  $K_i$  for which the ratio  $p_{K_i}/l_{K_i}$  is maximum.

The correctness of the last algorithm can be easily proved: it is enough to calculate how the sum will change if two items  $K_i$  and  $K_j$  change their positions in the final arrangement. We leave the immediate implementation to the reader.

## Genetic algorithms

- Genetic algorithms are a search-and-find method in which the basic idea is to simulate genetic and evolutionary processes in nature.
- Thus, for a given optimization problem, several random suboptimal solutions are initially constructed.
- The most successful ones are kept and new ones are built on their basis in the hope that they will be even better. In this way, suboptimal and unpromising solutions are isolated from further consideration.

# Genetic algorithms

- The analogy with the process of evolution of species is obvious - in nature this phrase is known as "survival of the fittest".
- Genetic algorithms are part of evolutionary programming, which is the rapidly growing field of artificial intelligence.
- The algorithm is run with a set of solutions (represented by chromosomes) called a **population**. Decisions from one population are taken and used by the new population.

# Genetic algorithms

- This is justified by the hope that the new **population** will be better than the old.
- The solutions that are chosen to form the new **population** (generation) are selected according to their viability - the more suitable ones have a greater chance of reproduction.
- This is repeated until some condition (e. g. the number of **populations** or proving the best solution) is satisfied.

*Genetic algorithms can be applied to a large class of problems, but a major disadvantage of the approach is the lack of theoretical evaluations guaranteeing good solutions.*

# Genetic algorithms

## Sketching the Genetic Algorithm

1. **[Start]** Generating a random population of  $n$  chromosomes (suitable problem solutions)
2. **[Viability]** Calculating the viability  $f(x)$  of each chromosome  $n$  in population
3. **[New population]** Creating a new population through repeating the following steps until the new population is finished
  1. **[Selection]** Selecting two parents chromosomes from the population according to their viability (better viability better selection chances)
  2. **[Crossbreeding]** By crossbreeding probably cross the parents to form the new generation (children). If no crossbreeding, the generation would be an exact copy of its parents.
  3. **[Mutation]** By mutation, probably the new generation is mutated at some place (place in the chromosome)
  4. **[Accepting]** Place the new generation in the new population
4. **[Replacement]** Using the new-generated population for further execution of the algorithm
5. **[Check]** If the final condition is satisfied, **stop**, and returning the best solution of current population.
6. **[Cycle]** Go to step 2

# Using Heuristics in Query Optimization

- **Process for heuristics optimization**
  1. The parser of a high-level query generates an initial internal representation;
  2. Apply heuristics rules to optimize the internal representation.
  3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.
- The main heuristic is to apply first the operations that reduce the size of intermediate results.
  - E.g., Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.

# Using Heuristics in Query Optimization

- **Query tree:**
  - A tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as **leaf nodes** of the **tree**, and represents the relational algebra operations as internal nodes.
- An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.
- **Query graph:**
  - A graph data structure that corresponds to a relational calculus expression. It does *not* indicate an order on which operations to perform first. There is only a *single* graph corresponding to each query.



# Questions and exercises:

1. Are heuristic algorithms exact algorithms?
2. When is it practical to use heuristic algorithms ?
3. What is the randomized algorithm ?



# Thank you for your attention!

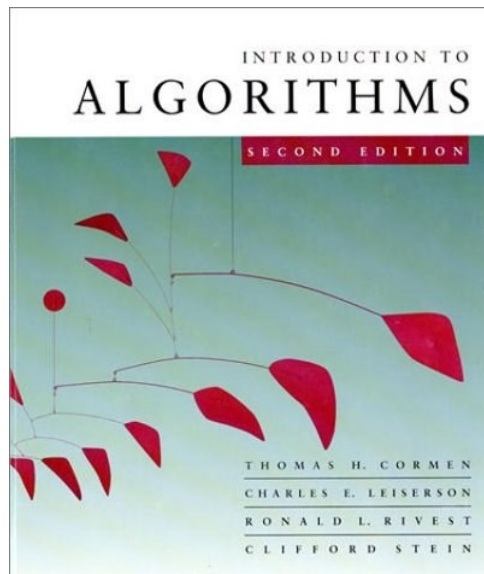
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 2. Strategies in Algorithm Design**
  - ❑ Lesson 4. Greedy algorithms. Examples.



ERASMUS+

# GREEDY ALGORITHMS. EXAMPLES.



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT  
Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov  
(2002). Programming= ++Algorithms). Top Team  
Co, София.

# Greedy Algorithms

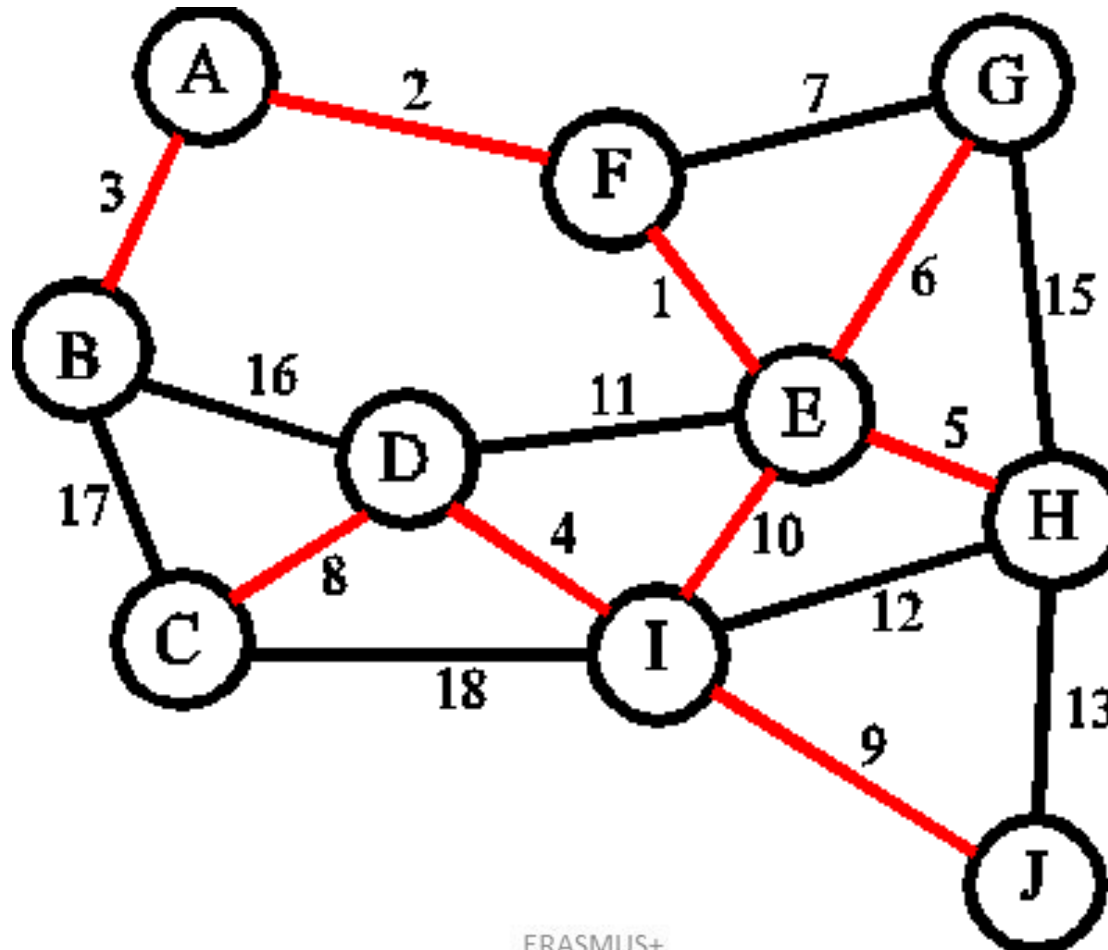
- Algorithms for optimization problems typically go through a series of steps, with a set of choices at each step.
- For many optimization problems, using dynamic programming to determine the best choice is unnecessary (too powerful weapon); simpler, more efficient algorithms can do this.
- A greedy algorithm always makes the choice that seems best at the time.
- That is, it makes a local optimal choice in the hope that this choice will lead to a global optimal solution.

# Greedy Algorithms

- **Greedy algorithms** do not always give optimal solutions, but for many problems they do find such a solution.
- In general, the greedy method is quite powerful and works well for a wide range of problems.
- Later (in the next lectures) we will present many algorithms where we may see applications of the greedy method including minimum covering trees, Djextra shortest path algorithm, etc.
- Algorithms for finding the maximum spanning tree represent a **classic example** of a **greedy algorithm**.

# Greedy Algorithms

## The cover tree



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness



# Greedy Algorithms

- **Greedy algorithms** are easy to compose, the corresponding implementation of the algorithm is not complicated, and the only disadvantage is that **sometimes they do not guarantee the correct solution of the problem.**
- However, it does not diminish their usefulness - it is characteristic of heuristic algorithms (and greedy ones in particular) that they quickly manage to find a near-optimal solution.
- In many practical problems, it is impossible to explore all cases, and often composing an algorithm that finds a solution 5% worse than the optimal solution is considered a success, compared to the alternative of an almost endless and unpromising search for the "true" optimal solution.

# Greedy Algorithms

- **The first problem** we will consider is the following: find a way to obtain a given sum  $m$  ( $m$  is a natural number), using a minimum number of banknotes, with denominations from the set  $C = \{a_1, a_2, \dots, a_n\}$ . For example, the banknotes are 1, 2, 5, 10, 20, 50 leva.
- *Consider the following algorithm:*
  - 1) Initialize  $s=0$
  - 2) Find the banknote  $i$  with a maximum value  $a_i$  ( $a_i \in C$ ) so that  $s+a_i \leq m$ .
    - 2.1) If there is not a banknote for which  $s+a_i \leq m$  the problem *has no solution*. The end.
    - 2.2) Otherwise we take the banknote  $i$  and increase  $s$  with  $a_i$ .
      - 2.2.1) If  $s=m$  the problem is solved. The end.
      - 2.2.2) If  $s < m$  than we still haven't obtained the total sum and repeat step 2).
- For example, for the sum of 298 leva, will be chosen sequentially five 50 leva notes, two 20 leva notes, one 5 leva note and one 2 leva note and one 1 leva note (total  $250 + 40 + 5 + 2 + 1 = 298$ ).

# Greedy Algorithms

- Obviously, the described algorithm satisfies the criteria of a greedy algorithm: at each step, it selects the maximum-valued banknote, thus aiming to achieve the searched amount as fast as possible. In this particular example, it leads to an efficient solution of the problem.
- For example, consider the case where the possible values for notes are 2, 5, 20 and 30 and we want to get an amount of 40.
- The greedy algorithm will first select a note of **30** (the maximum possible), then select **two** notes of **5**, i. e. **3 notes in total**.

# Greedy Algorithms

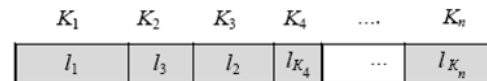
- Obviously, however, there is a better solution: **two 20 notes**.
- In addition to not finding the optimal solution, it is possible that the greedy algorithm may **not find** a solution at all.
- So, for example, if we want to get a sum of 6: after the note with a value **5**, the algorithm is left with no further choices (and the sum can still be obtained from **3 notes** with a value **2**).
- However, things are not always so bad and there are a number of problems where it can be shown that the *greedy algorithm* always finds a solution.

# Greedy Algorithms

## Magnetic stripe problem

Given  $n$  programs of lengths  $l_1, l_2, \dots, l_n$  and a magnetic stripe with a sequential access.

- Sequential access means that in order to read a record located at a given position on the stripe, you must pass (scroll the stripe) to the specified location.



Magnetic stripe with a sequential access ( $K_1 = 1, K_2 = 3, K_3 = 2, \dots$ ).

- For example, in order to read the third program (of length  $l_2$ ), all programs before it - those of length  $l_1$  and  $l_3$  - must be "scrolled".

# Greedy Algorithms

## Magnetic stripe problem

Given the probabilities of execution of each program  $p_1, p_2, \dots, p_n$ ,  $0 \leq p_i \leq 1, 1 \leq i \leq n$ ,  $\sum_{i=1}^n p_i = 1$

For example a probability 0.5 means the program is running as often as all the other programs together. In fixed ordering  $K_1, K_2, \dots, K_n$  of the programs on the stripe the *average time*  $T$  needed for loading a random program is defined as follows:

$$T = \frac{\sum_{i=1}^n \left( p_{K_i} \sum_{j=1}^i l_{K_j} \right)}{n}$$

The task is to find an ordering of the programs on the stripe minimizing  $T$ .

We are going to use a greedy algorithm. It is clear that as often a program is used further on the stripe it should be written. The other defining criterion is the length of the stripe – as longer it is as backwards on the stripe it should be (to “hinder” less when it needs to be scrolled).

Thus, we reach the following algorithm: Write sequentially the programs on the stripe and on each step we choose program  $K_i$  for which the ratio  $p_{K_i}/l_{K_i}$  is maximum.

The correctness of the last algorithm can be easily proved: it is sufficient to calculate how the sum will change if two members  $K_i$  and  $K_j$  change their positions in the final ordering.

# Greedy Algorithms

## Maximum combination of activities

*Problem:* Given  $n$  lectures to be taught (activities to be performed). Each lecture  $i$  is defined by two numbers: a fixed start  $s_i$  and a fixed end  $f_i$ .

The numbers  $s_i$  and  $f_i$  can denote the start and end time for the lecture - we can consider them to be natural numbers.

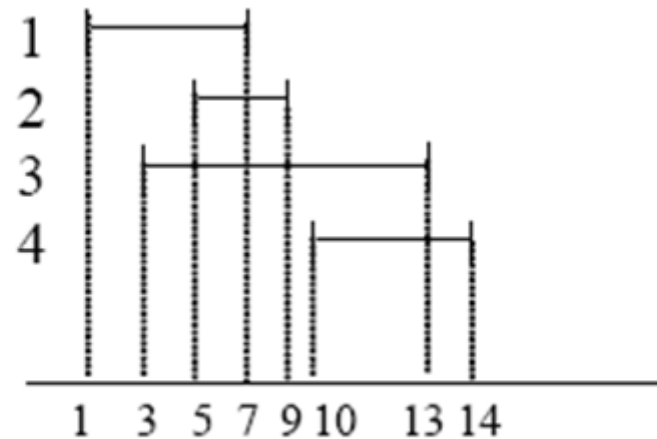
A maximum number of lectures should be chosen so that no two of the chosen lectures are given at the same time, i. e., assuming that we have only one lecture hall, then at any time  $t$  at most one lecture can be given in the hall

$i (s_i \leq t \leq f_i, 1 \leq i \leq n).$



# Greedy Algorithms

## Maximum combination of activities



Configuration of four lectures: beginning and end.

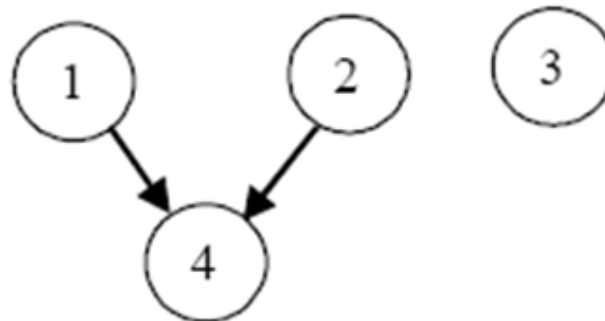
The problem is a classic example of an efficient and correctly working greedy algorithm . Lectures numbered 1 and 4 do not intersect in time, while 1 and 2, 2 and 3, etc. intersect (i. e. cannot be held simultaneously).

# Greedy Algorithms

## Maximum combination of activities

*Algorithm 1:*

we construct an oriented graph  $G(V, E)$  with  $n$  vertices in which every two “non-intersecting” lectures  $i, j$  ( $f_i < s_j$ ) corresponds to an edge  $(i, j)$  of the graph. If we find the longest path in  $G$  we get an algorithm with complexity  $\Theta(n^2)$ .



**Configuration graph**

# Greedy Algorithms

## Maximum combination of activities

### *Algorithm 2:*

It is possible to solve the problem without constructing a graph by using dynamic programming. We define a target function  $F$ , maximizing the number of the lectures chosen for time from 0 to  $t$  the following way:

$F(t) = 0$ , if no lecture  $i$  exists for which  $f_i < t$  and

$F(t) = \max_{\substack{i=1,2,\dots,n \\ f_i < t}} \{F(s_i) + 1\}$  otherwise

The solution is the value of function  $F(t_0)$  for  $t_0 = \max_{i=1,2,\dots,n} \{f_i\}$

# Greedy Algorithms

## Maximum combination of activities

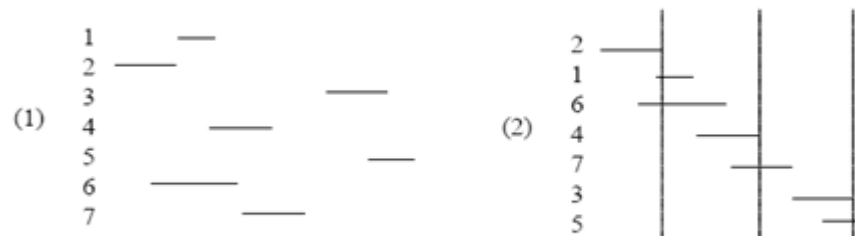
### Algorithm 3:

The problem can be solved simply and effectively using greedy algorithm, where a complexity  $\Theta(n \cdot \log_2 n)$  is achieved without using additional memory. Complexity of greedy algorithm is even linear – the problem is that we should have the lectures sorted in ascending order by  $f_i$ . And because sorting in general case is with complexity  $\Theta(n \cdot \log_2 n)$  it defines the complexity of solution itself.

### Algorithm:

We consider the lectures sequentially from first to last:

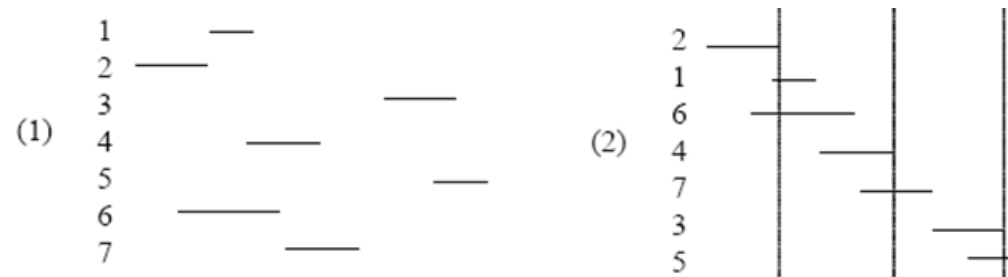
- 1) We choose lecture  $i$  for which  $f_i$  is minimal (at the beginning always choose the first one because lectures are sorted in ascending order by  $f_i$ )
- 2) All lectures  $j$  for which  $s_j \leq f_i$  (i.e. which “interfere” with the selected one) are excluded from the lectures list. Step 1 is repeated and so until all lectures are taught.



Sorting the lectures and finding the maximum combination

# Greedy Algorithms

## Maximum combination of activities



Choosing the lectures and find the maximum combination

Fig. (2) shows the lectures from (1) sorted by  $f_i$ . The greedy algorithm will choose lecture 2, then will skip lectures 1 and 6, and choose 4, skip 7, choose 3 and skip 5. The algorithm scheme as a pseudocode looks generally like this:

```
Choose_lecture_1
i = 1; j = 1;
while (j <= n) {
    j++;
    if (s[j] > f[i]) {
        Choose_lecture_j
        i = j;
    }
}
```

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# Greedy Algorithms

## Maximum combination of activities

follows the source code of the program (we assume that the lectures are already sorted by  $f_i$ ).  
The input data are set at the beginning as constants.

```
#include <stdio.h>
#define MAXN 100

const unsigned n = 7;
const unsigned s[MAXN] = { 3, 7, 5, 9, 13, 15, 17 };
const unsigned f[MAXN] = { 8, 10, 12, 14, 15, 19, 20 };

void solve(void)
{ unsigned i = 1, j = 1;
  printf(" selected lectures : %u ", 1);

  while (j++ <= n)
    if (s[j - 1] > f[i - 1]) printf("%u ", i = j);
  printf("\n");
}

int main(void) {
  solve();
  return 0;
}
```

# Greedy Algorithms

## *Elements of the greedy strategy:*

- A greedy algorithm obtains an optimal solution to a problem by making a sequence of choices. At each decision point, the algorithm makes choice that seems best at the moment. This heuristic strategy does not always produce an optimal solution, but as we saw in the activity-selection problem, sometimes it does.
- More generally, we design greedy algorithms according to the following sequence of steps:
  1. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.



# Greedy Algorithms

## *Elements of the greedy strategy:*

- More generally, we design greedy algorithms according to the following sequence of steps:
  2. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
  3. Demonstrate optimal substructure by showing that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem
- Every greedy algorithm, there is almost always a more cumbersome dynamic-programming solution.

# Greedy Algorithms

## *Greedy-choice property:*

- The first key ingredient is the greedy-choice property: we can assemble a globally optimal solution by making locally optimal (greedy) choices.
- In other words, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems.

# Greedy Algorithms

## *Greedy-choice property:*

- **Here is where greedy algorithms differ from dynamic programming.** In dynamic programming, we make a choice at each step, but the choice usually depends on the solutions to subproblems.
- Consequently, we typically solve dynamic-programming problems in a bottom-up manner, progressing from smaller subproblems to larger subproblems. (Alternatively, we can solve them top down, but memorizing.
- Of course, even though the code works top down, we still must solve the subproblems before making a choice.) In a greedy algorithm, we make whatever choice seems best at the moment and then solve the subproblem that remains.

# Greedy Algorithms

## *Greedy-choice property:*

- The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to subproblems.
- Thus, unlike dynamic programming, which solves the subproblems before making the first choice, a greedy algorithm makes its first choice before solving any subproblems.
- A dynamic programming algorithm proceeds bottom up, whereas a greedy strategy usually progresses in a top-down fashion, making one greedy choice after another, reducing each given problem instance to a smaller one.

# Greedy Algorithms

## *Summary:*

- ❖ Any algorithm that is implemented by selecting (locally) at each step the maximum (or in some cases the minimum, depending on what the algorithm wants to achieve) element of a given set (i. e. the relation  $>$  in this set is introduced), "hoping" that this will lead to the global maximum is called a Greedy algorithm. It is this "take what you can get at the time" strategy that gives the name to this class of algorithms.
- ❖ They are simple and "one-way" - they make decisions based on the information we have now without worrying about what will happen in the future.

# Greedy Algorithms

## *Summary:*

- ❖ Greedy algorithms never review the decisions they make. This distinguishes them from dynamic optimization, where the decision that is made is based on all possible decisions made so far - so we are sure that our algorithm outputs the optimal one.
- ❖ When running of the algorithm is finished, if the maximum we have obtained matches the global maximum (i. e. the best solution) then our algorithm is correct otherwise the greedy algorithm outputs an answer very close to the optimal one.
- ❖ Greedy algorithms are easy to implement, unlike dynamic optimization algorithms, and in most cases quite efficient. These properties make them a good choice for solving competitive problems.
- ❖ **Query optimizers can use a faster method called Greedy for determining the order in which to join the tables referenced.**

# Questions and exercises:

1. What is a Greedy Algorithm ?
2. What Are Greedy Algorithms Used For ?
3. What is the difference between Dynamic Programming and Greedy Algorithms ?





# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 3. Algorithms and their applications in databases for query optimization

### ❑ Topic 3. Sorting algorithms

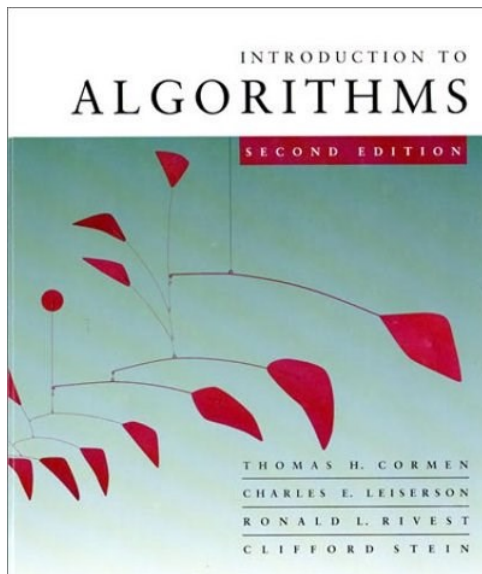
- ❑ Lesson 1. Sorting algorithms - part I (Sorting by insertion.  
Sorting by direct selection. Bubble method).



ERASMUS+

# Sorting Part I

Sorting algorithms. Insertion sort. Selection sort. Bubble method.



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov (2002).  
Programming = ++Algorithms). Top Team Co, София.

# Sorting

- Often, when working with large data of the same type, it is necessary to introduce some kind of ordinance in order to process them more easily.
- The ordering of the items could give us a significantly more efficient search algorithm compared to the case where the data is not ordered.
- It is customary to call the process of rearranging (permuting appropriately) the elements of some set of objects in a particular order a sorting.

# Sorting

- Sorting is one of the primary algorithms used in query processing. For example, whenever an **SQL query specifies** an ORDER BY-clause, the query result must be sorted.
- Sorting is also a key component in sort-merge algorithms used for JOIN and other operations (such as UNION and INTERSECTION), and in duplicate elimination algorithms for the PROJECT operation (when an SQL query specifies the DISTINCT option in the SELECT clause).

# Sorting

- The variety of algorithms is so great that Knuth devoted the entire third volume (over 800 pages) of his famous monograph “The Art of Computer Programming” to sorting (and searching).
- Niklaus Wirth, the creator of the Pascal language, also devotes quite a bit of attention to them in Algorithms + Data Structures = Programs.
- The process of sorting the elements of a set, as already mentioned above, generally comes down to rearranging them. We will define the terms more strictly.

# Sorting

- Let the set  $M$  be given with elements:

$$a_1, a_2, \dots, a_n$$

and a function  $f$  defined on them.

- By sorting the elements of  $M$  we mean permuting them in the appropriate order :

$$a_{i_1}, a_{i_2}, \dots, a_{i_n}$$

so that  $f(a_{i_1}) \leq f(a_{i_2}) \leq \dots \leq f(a_{i_n})$ . is fulfilled.

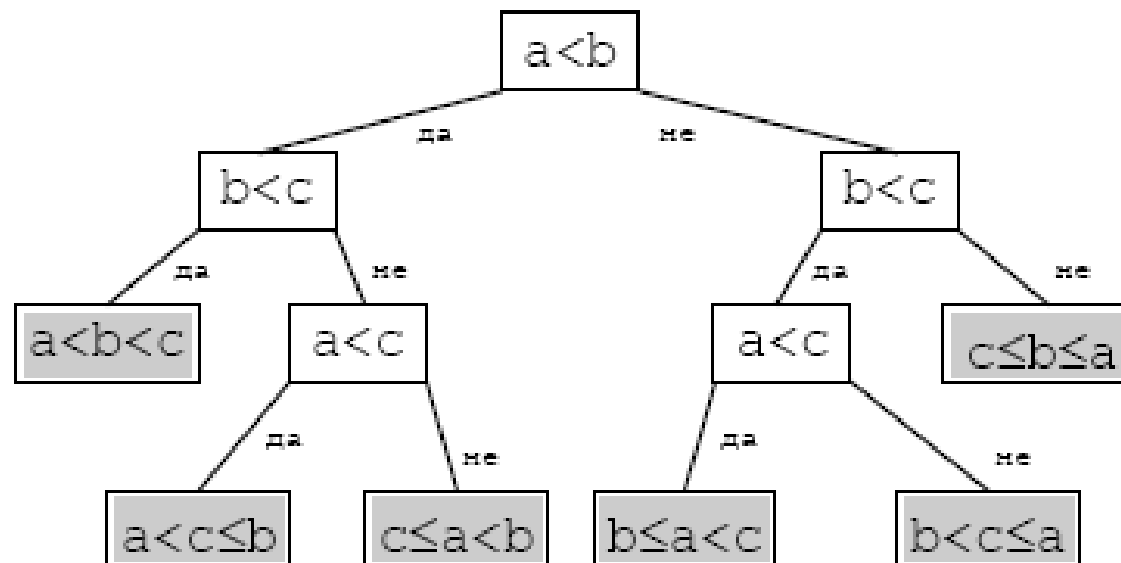
- The function  $f$  is called a set ordering function.



# Comparison sorting

- These are *classical algorithms* where the only allowed operation is a comparison between pairs of elements using the operations  $< (\leq)$ ,  $> (\geq)$  и  $= (\neq)$ .

## Tree of comparisons for the set $\{a, b, c\}$



# Comparison sorting

- This is probably the most elementary method of sorting by comparison. It is based on a series of comparisons, each of which brings us additional information.
- The process continues until the set is completely sorted.
- We may consider that any comparison of the form  $x < y$  has two outcomes: yes if  $x$  is less than  $y$ , and no otherwise.
- We're going to illustrate the method on the three-element set  $\{a, b, c\}$ .

# Comparison sorting

```
if (a < b)
    if (b < c) printf("a,b,c");
    else
        if (a < c) printf("a,c,b");
        else printf("c,a,b");
else
    if (b < c)
        if (a < c) printf("b,a,c");
        else printf("b,c,a");
    else printf("c,b,a");
```

- The above algorithm is beautiful and useful insofar as:
  - 1) guarantees a minimum number of comparisons;
  - 2) gives obviously the tree of comparisons (used to prove minimum time complexity).

# Comparison sorting

- Its advantages, unfortunately, end here.
- It is not hard to notice that the number of possible outputs (leaves of the tree) is  $n!$ , as many as there are possible permutations of the elements of the output set, which, even with a sufficiently small number of elements, is a rather large number, not allowing to write a relevant program.
- Other more efficient sorting methods have to be sought.

# Insertion sorting

- There are three classical elementary universal methods for sorting **by comparison**: by insertion, by selection and by the bubble method.
- Elementary sorting methods are effective for relatively small numbers of items (**about 20**) and are often used in practice.
- Unfortunately, with a larger number of elements, their speed slows sharply, so other methods have to be used.
- Indeed, all three elementary methods are characterized by an algorithmic complexity  $O(n^2)$ , which is much slower compared to the complexity  $O(n \log_2 n)$  characteristic of modern sorting methods such as *pyramidal* or *fast sorting*.

# Insertion sorting

- Insertion sorting is the well-known method of ordering cards, where the player, holding the cards in his left hand, removes them one by one into his right hand and places them in their correct position.
- Inserting the card in the correct position requires comparing it more consistently (by eye) with the cards already stacked until the correct position is found.
- How is the insertion done? One obvious algorithm is the sequential comparison and eventual exchange of  $x$  with an element to its left. The process continues until one of the following situations occurs:
  - 1) reaching an element with a key less than or equal to  $x$ ;
  - 2) reaching the first element of the array.

# Insertion sorting

The method implements the following idea.

If the row

$a_0, a_1, \dots, a_{i-1}$

is sorted in ascending order, the element  $a_i$  is included in the row so that it takes the "correct" position, i. e. the new row

$a_0, a_1, \dots, a_i, \dots, a_{i-1}$

is sorted in ascending order. These actions are repeated for  $i = 1, 2, \dots, n-1$



# Insertion sorting

## *Source code of insertion sorting algorithm of C++ language*

*// A procedure that inserts elements in a sorted in ascending order row*

```
void Insert(int a[], int i)
{
    int j;
    int x = a[i];
    for(j = i - 1; j >= 0; j --)
        if(x < a[j])
            a[j+1] = a[j];
        else
            break;
    a[j+1] = x;
}
```

# Insertion sorting

**EXAMPLE:** Insertion sort.

491	543	632	183	687	632
-----	-----	-----	-----	-----	-----

Initial array

491	543	632	183	687	632
-----	-----	-----	-----	-----	-----

543 keeps its position

491	543	632	183	687	632
-----	-----	-----	-----	-----	-----

632 keeps its position

491	543	632	183	687	632
-----	-----	-----	-----	-----	-----

The place of 183 in the sorted part of the list is before 491

183	491	543	632	687	632
-----	-----	-----	-----	-----	-----

687 keeps its position

183	491	543	632	687	632
-----	-----	-----	-----	-----	-----

The place of 632 in the sorted part of the list is before 687

183	491	543	632	632	687
-----	-----	-----	-----	-----	-----

Sorted array

# Sorting by direct selection method

- The idea of the method is as follows: the smallest element is searched for, moved to the beginning of the array and excluded from consideration.
- The operation is repeated until all  $n$  elements are selected.

For  $i = 0, 1, \dots, n-2$  the sequence is considered :

$a_i, a_{i+1}, \dots, a_{n-1}$

and the following actions are performed:

- Find  $k$  , so that  $a_k = \min\{a_i, a_{i+1}, \dots, a_{n-1}\}$ .
- exchange the values of  $a_{i+1}$  and  $a_i$ .

# Sorting by direct selection method

function implementing sorting by the method of direct selection

```
void SelectSort(int a[], int n)
{
    int min, k;
    for(int i = 0; i < n; i ++)
    {
        k = i;
        min = a[k];
        for(int j = i + 1; j < n; j ++)
            if(a[j] < min)
            {
                k = j;
                min = a[k];
            }
        a[k] = a[i];
        a[i] = min;
    }
}
```

# Sorting by direct selection method

## EXAMPLE: Sorting by direct selection method

491	543	632	183	678	632	Initial array
183	543	632	491	678	632	exchange 491 and 183 (the minimum in the unsorted part)
183	491	632	543	678	632	exchange 543 and 491 (the minimum in the unsorted part)
183	491	543	632	678	632	exchange 632 and 543 (the minimum in the unsorted part)
183	491	543	632	678	632	no exchanges are made
183	491	543	632	632	678	exchange 678 and 632 (the minimum in the unsorted part)
183	491	543	632	632	678	Sorted array

# BUBBLE SORT METHOD

It is recommended for a comparatively small  $n$

We are sorting only in ascending row:

$a_0, a_1, \dots, a_{n-1}$

The method can be explained the following way:

1. Let **right** =  $n-1$
2. For the row

$a_0, a_1, \dots, a_{\text{right}}$

consistently compare any two adjacent elements  $a_i$  and  $a_{i+1}$

If  $a_i > a_{i+1}$ , both elements exchange their places and the  $i$  position of exchange is saved in the variable **k**.

# BUBBLE SORT METHOD

If  $a_i \leq a_{i+1}$  the two elements do not exchange places.

The process continues until the end of the row. If in this scanning after the exchange of elements from position  $i$  and  $i+1$  no other exchange is made, the elements of positions:  $k+1, k+2, \dots, n-1$  are properly arranged.

3. **right** =  $k$

4. If **right** > 0 the actions from p. 2 and p. 3 repeat.

5. If **right** = 0 the row is sorted.



# Bubble sort method

*Function implementing sorting by the bubble method of the C++ language*

```
void BubbleSort(int a[], int n)
{
    int right = n-1;
    int temp = 0;
    while(right > 0)
    {
        int k = 0;
        for(int i = 0; i < right; i++)
            if(a[i+1] < a[i])
            {
                temp = a[i+1];
                a[i+1] = a[i];
                a[i] = temp;
                k = i;
            }
        right = k;
    }
}
```

# Bubble sort method

## EXAMPLE: Bubble sort method

491 543 632 183 678 362 **Initial array**

491 543 632 183 678 362 543 > 491, no exchange is made

491 543 632 183 678 362 632 > 543, no exchange is made

491 543 632 183 678 362 183 < 632, an exchange is made

491 543 183 632 678 362 678 > 632, no exchange is made

491 543 183 632 678 362 362 < 678, an exchange is made

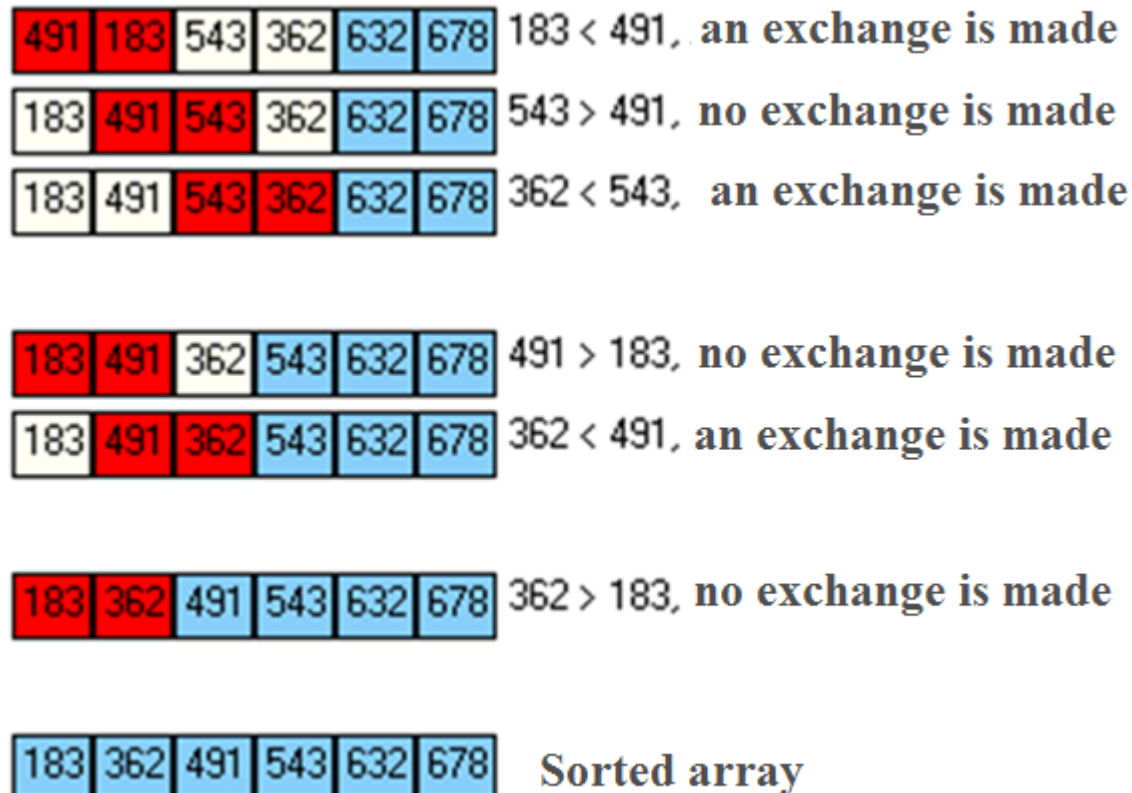
491 543 183 632 362 678 543 > 491, no exchange is made

491 543 183 632 362 678 183 < 543, an exchange is made

491 183 543 632 362 678 632 > 543, no exchange is made

491 183 543 632 362 678 362 < 632, an exchange is made

# Bubble sort method



<i>Name of the algorithm</i>	<i>Average case time complexity</i>	<i>Worst case time complexity</i>	<i>Stable?</i>
Bubble sort	$\Theta(n^2)$	$O(n^2)$	Yes
Selection sort	$\Theta(n^2)$	$O(n^2)$	No
Insertion sort	$\Theta(n^2)$	$O(n^2)$	Yes

Selection sort works by finding the minimal element and then inserting it into its correct position by swapping with the element that is in that minimal element's position. This makes it unstable.



Algorithm	Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)	Space Complexity
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$

Since the sorting algorithms discussed so far **are efficient only with small input data size, with large data volume it is necessary to use algorithms with the  $O(n \log_2 n)$  complexity** typical of sorting methods such as merge sort, heap sort or quick sort.

Since the sorting algorithms examined so far are only effective at a small size of input data, with a large volume of data, it is necessary to use algorithms with the complexity  $O(n \cdot \log_2 n)$  characteristic of sorting methods such as merge sorting, heap or quick sorting.

## Questions and exercises:

1. Why Sorting algorithms are important ?
2. Which of the three sorting methods discussed is unstable and why ?
3. Explain how bubble sort works ?
4. Explain how insertion sort works ?





# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 3. Algorithms and their applications in databases for query optimization

### ☐ Topic 3. Sorting algorithms

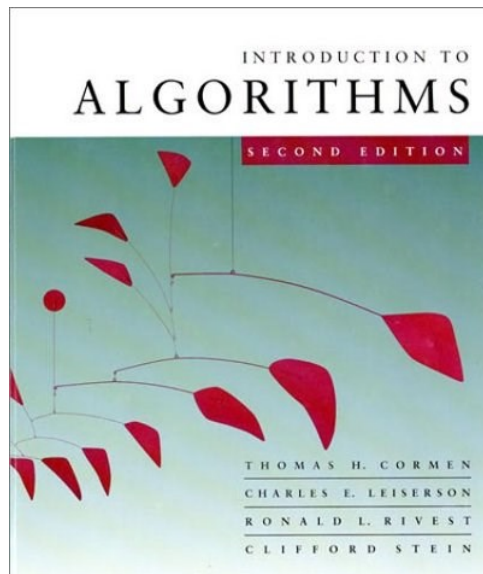
- ☐ Lesson 2. Sorting algorithms - part II (Linear time sort - Quick sort. Merge sort. Heap sort.)



ERASMUS+

## SORTING PART II

Linear time sort. Quick sort. Merge sort. Heap sort.



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT  
Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov  
(2002). Programming= ++Algorithms). Top Team  
Co, Sofia.

# Sorting

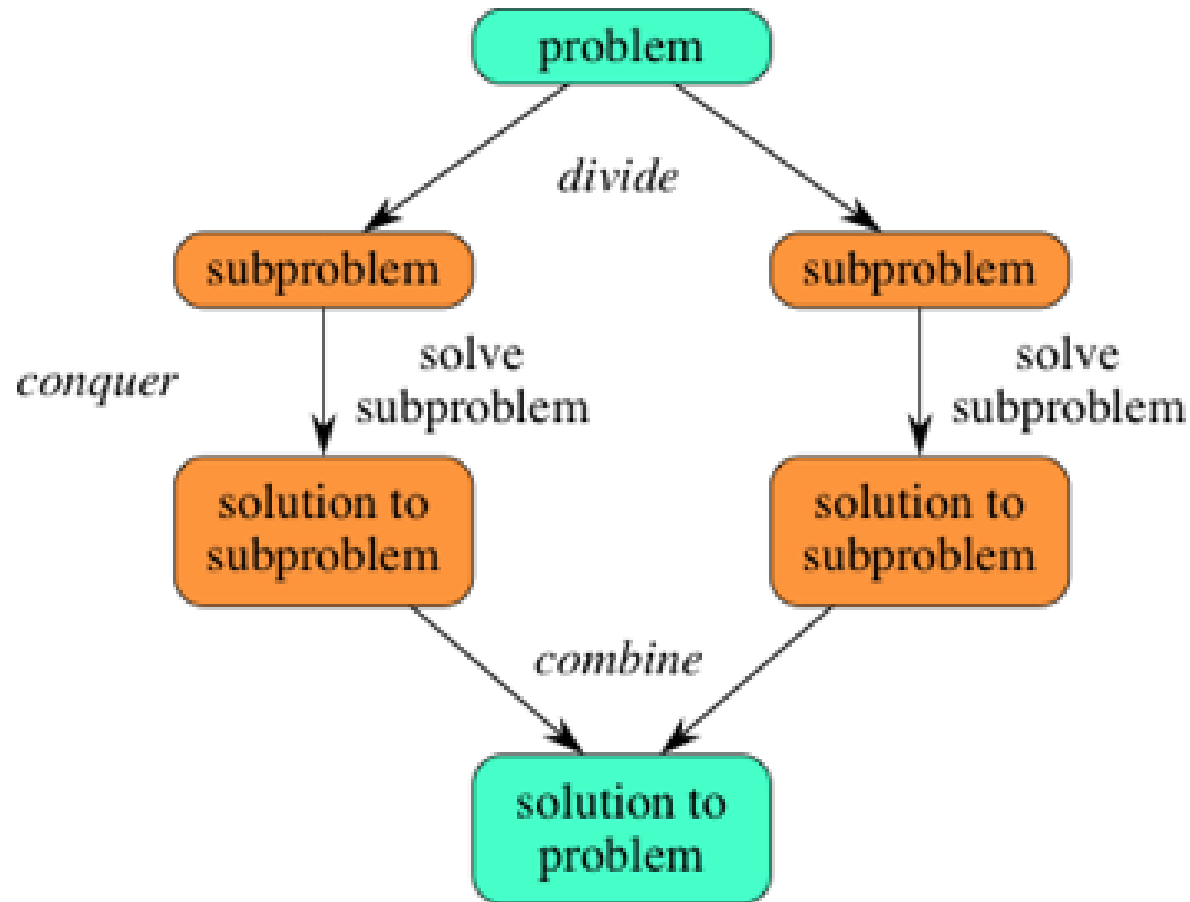
- In the previous lecture, we introduced three algorithms based on sorting by comparison: by insertion, by selection, and by the bubble method.
- These algorithms have complexity  $O(n^2)$  in the worst case, some of them also in the average case.
- However, these algorithms are fast for small input sizes.

# Quicksort

- Proposed by C.A.R. Hoare in 1962
- **Divide and conquer algorithm.**
- Sort "in-place" (like insert sorting, but different from merge sorting).
- It has a lots of practical applications (with various modifications).

# Quicksort

- Divide and conquer algorithm (revision).







# Quicksort

- Quicksort of an array with  $n$  elements:
  1. **Divide:** split the array into two subarrays around center  $x$ , the elements in the lower subarray  $\leq x \leq$  by the elements in the upper subarray.



2. **Rule:** Recursive sorting of the two subarrays.
3. **Combine:** trivial.

Keyword: linear time for subprograms splitting

# Quicksort

- We apply the same algorithm to the left and right parts, gradually reducing the left and right boundaries of the considered subarrays until we reach intervals containing a single element.
- After the algorithm finishes its work, the array will be sorted.
- In the worst case, the algorithm has complexity  $O(n^2)$ .
- In the average case, the algorithm has complexity  $O(n \log n)$

For example  $n=100$ ,  $n^2=10\,000$ ,  $\log n=6.64$ ,  $n \cdot \log n=664$

# Quicksort

## *Action principal:*

1. Select a “**pivot**” element from the list of items to be sorted.
2. The list is rearranged so that all items that are smaller than the “**pivot**” element are placed to the left of it, and all items that are larger are placed to the right of it.
3. Recursively repeat the above steps on the list with the smaller and the list with the larger elements.
4. The resulting lists are merged (concatenation) to produce the sorted list.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# Quicksort

## *Example 1:*

[5 3 2 8 7 6 1 9 4]

For a "pivot" element choose **5**. Divide the list into three parts:  
elements smaller than "pivot", "pivot", bigger than "pivot"

[3 2 1 4] | [5] | [8 7 6 9]

Recursively perform the actions on the newly obtained list

[2 1] | [3] | [4] | [5] | [7 6] | [8] | [9]

[1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9]

Link the lists and obtain the sorted list

[1 2 3 4 5 6 7 8 9]

# Quicksort

*Using a simplified pseudocode, we can see the implementation of algorithm:*

```
1  function quicksort('array')
2      if length('array') ≤ 1
3          return 'array' // an array of zero or one elements is already
sorted
4      select and remove a pivot element 'pivot' from 'array' //
see selection of "pivot" element below
5      create empty lists 'less' and 'greater'
6      for each 'x' in 'array'
7          if 'x' ≤ 'pivot' then append 'x' to 'less'
8          else append 'x' to 'greater'
9      return concatenate(quicksort('less'), list('pivot'),
quicksort('greater')) // two recursive requirements
```

# Quicksort

- We can see that the elements are checked only by comparing them with other elements. This describes quick sorting as sorting by comparison.
- The correctness of the algorithm is based on the following two arguments:
  - at each iteration, the already processed elements are placed at the desired position: before the “pivot” element if they are smaller than it and after the “pivot” element if they are larger than it;
  - Each iteration decreases the number of elements that are not yet ordered by one.

***The correctness of the whole algorithm is proved by induction: for zero or one element, the algorithm leaves the data unchanged; for a larger dataset, the algorithm concatenates two parts - elements smaller than the “pivot” and elements larger than it.***



# Quicksort

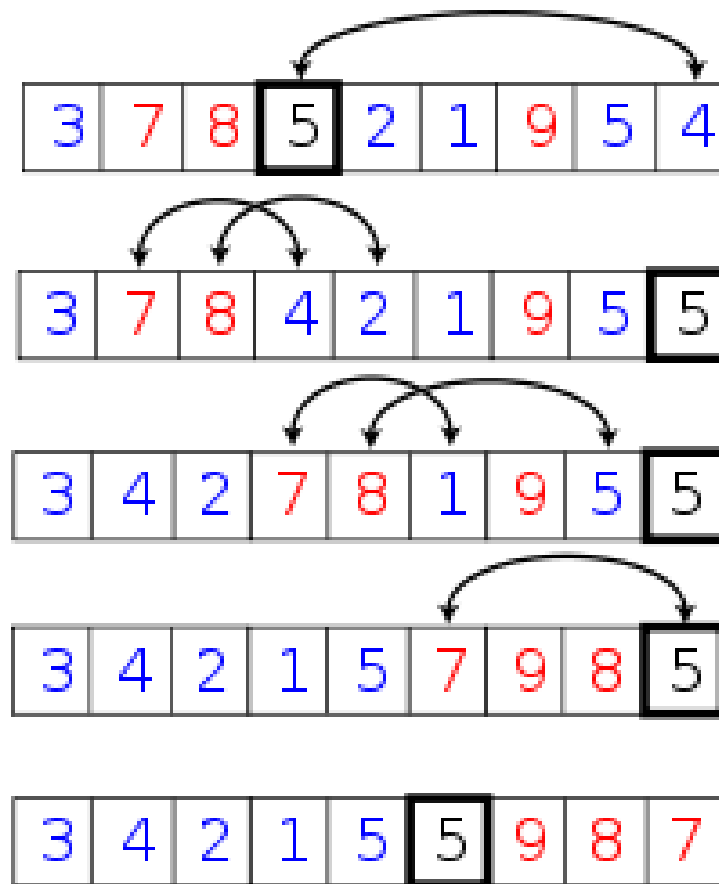
*Interactive example (Example 2) for a Quicksort:*

6 5 3 1 8 7 2 4



# Quicksort

*Example 3 for a Quicksort:*



The marked element is the pivot element, blue elements are smaller or equal, and red elements are larger.

# Quicksort

## *Selection of pivot element:*

- In the earliest versions of quick sort, the leftmost item was often selected as **the pivot**.
- Unfortunately, this doesn't work for already sorted arrays, which are a fairly common case.
- The problem was easily solved by choosing either an arbitrary index for the pivot element, or the middle element of the partition, or (especially for longer series) the median of the first, middle and last elements of the partition (as recommended by Robert Sedgewick).

# Quicksort

- In the quick sort, the elements are not inserted sequentially into a particular tree structure, but simultaneously form a tree, which is obtained in the recursive call.
- The closest competitor to quick sort is *heap sort*.
- Here the worst time is always  $O(\log n)$ , but heap sorting is generally considered slower than standard fast sorting.
- **Merge sort  $O(\log n)$** . is another recursive algorithm that is comparable to quick sort, with a worst-case time of  **$O(n \log n)$** .

# Mergesort

- At each step of the merge, we compare one element from the first half of  $a$  to one element from the second half of  $a$  and write one element to  $b$ , i. e. , we make 3 calls to elements of the arrays.
- It is a stable algorithm in contrast to fast sorting in-place and heap sorting, and can be easily adapted for use in linked lists and very large lists stored on slow cartridge such as disks or network storage.
- The main disadvantage of merge sort is the need for  $O(n)$  additional array processing **space**, while the basic variant of fast in-place sort and queue recursion use only  $O(\log n)$ .

# Mergesort

- The algorithm is based on the principle of “divide and conquer”.
- Principle of action:
  1. The unsorted list is randomly divided into two sublists of approximately total length (for linear time);
  2. Recursively split sublists until single-length lists are reached;
  3. Merge two sublists into a new sorted list (for linear time).
- The advantage of merge sorting is that it **always works with complexity  $n*\log(n)$** .

# Mergesort

- The basis of this sorting algorithm is the “**divide-and-conquer**” method: the sorted sequence is split into two parts, each of which is sorted, and then the sorted subarrays are merged.
- The merge sort algorithm is one of the most **efficient** sorting algorithms known.
- The algorithm can be used in parallel, i. e. in each core of the processor is fed the method that splits the array into 2, and then the numbers are merged at once.
- If we have 8 microprocessors, each processor is given the Split method, and then all 8 parts are joined at once.
- The algorithm can become  $n$  times faster when we have  $n$  processors

# Mergesort

*Source code of the merge sort algorithm of C++ language*

*// A procedure that implements merging of two arrays into a third one*

```
void merge(const int a[], int n, const int b[], int m, int* c)
{
    int i = 0, j = 0, k = -1;

    while(i < n && j < m)
        if(a[i] < b[j])
        {
            k++;
            c[k] = a[i];
            i++;
        }
        else
        {
            k++;
            c[k] = b[j];
            j++;
        }

    if(i == n)
        while(j < m)
        {
            k++;
            c[k] = b[j];
            j++;
        }
    else
        while(i < n)
        {
            k++;
            c[k] = a[i];
            i++;
        }
}
```



# Mergesort

*// The procedure that solves the merge sort task*

```
void MergeSort(int a[], int n)
{
    if(n < 2)
        return;

    int nLeft = n / 2;
    int nRight = n - nLeft;

    MergeSort(a, nLeft);
    MergeSort(a + nLeft, nRight);

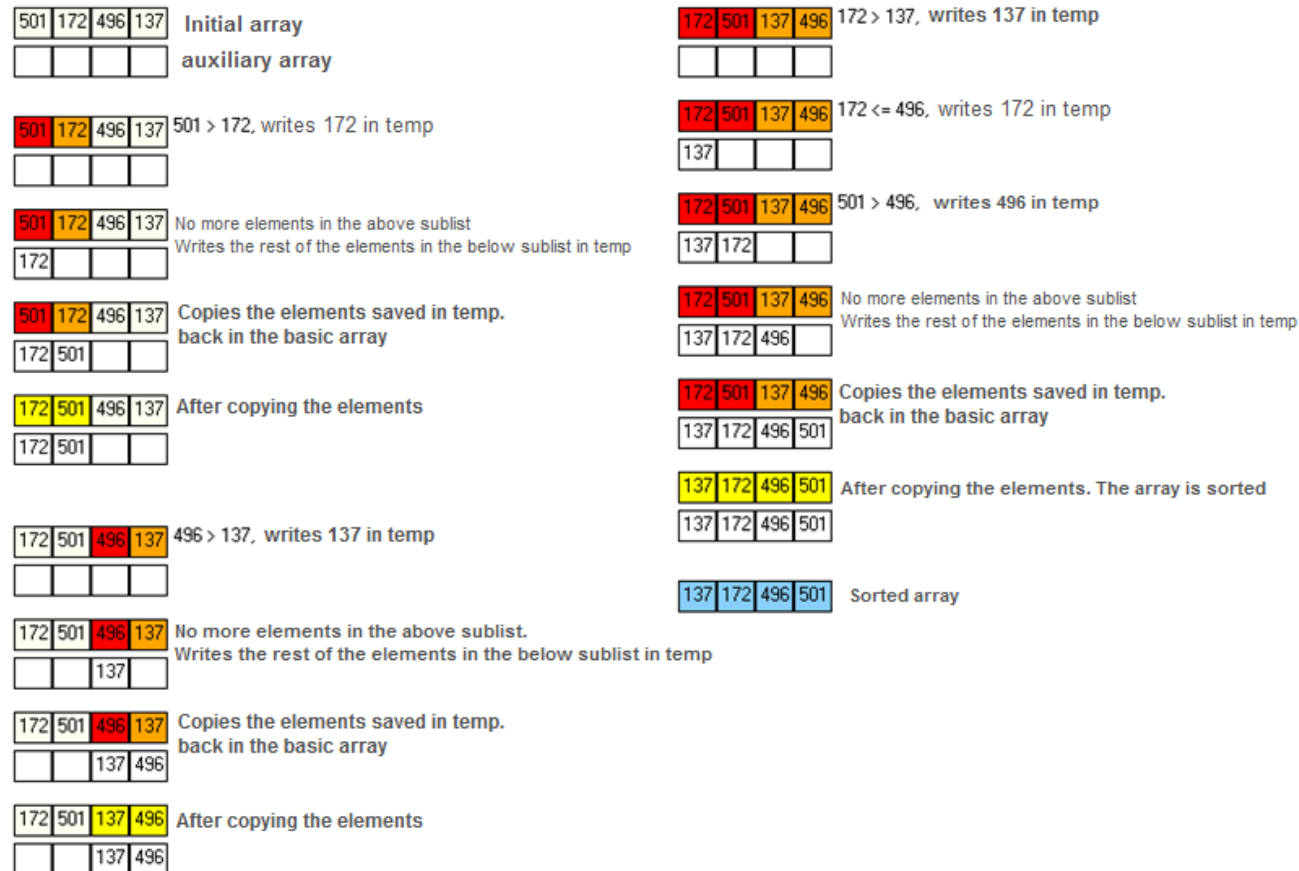
    int* p = new int[n];
    merge(a, nLeft, a + nLeft, nRight, p);

    for(int i = 0; i < n; i++)
        a[i] = p[i];

    delete [] p;
}
```

# Mergesort

## Example: Merge sort.



# Mergesort

*An interactive example of merge sort.*

6 5 3 1 8 7 2 4

# Heapsort

- Heapsort is a type of sort algorithm by direct selection.
- Although it is slower, with complexity  $O(n \log n)$ , than a good application of the quicksort algorithm, it has the advantage of a more favorable worst case because it does not use many arrays or recursion.
- Heapsort was invented by J. W. J. Williams in 1964.
- Heap sort can be divided into two steps:
  - First step, create a "**heap**" of elements of the set;
  - Step two, a sorted array is created by taking and removing the largest element from a pyramidal data structure (heap) and placing it in another array.

# Heapsort

- The "heap" is reconstructed on each remove, and once all objects have been removed from the heap memory, the full sorted array is created.
- The direction of the sorted items can be changed by selecting min-heap or max-heap in the first step.
- to execute this algorithm two arrays are required - one for the elements in the heap, a second for the sorted array.
- Heap sort can also be done with a single array - when an element is removed from the "heap", it makes space, and the element can be added to the end of the same array.
- Heapsort is mainly competed by quicksort. The worst case of quicksort is  $O(n^2)$ .

# Heapsort

*Interactive example for a heapsort*

6 5 3 1 8 7 2 4

# Sorting

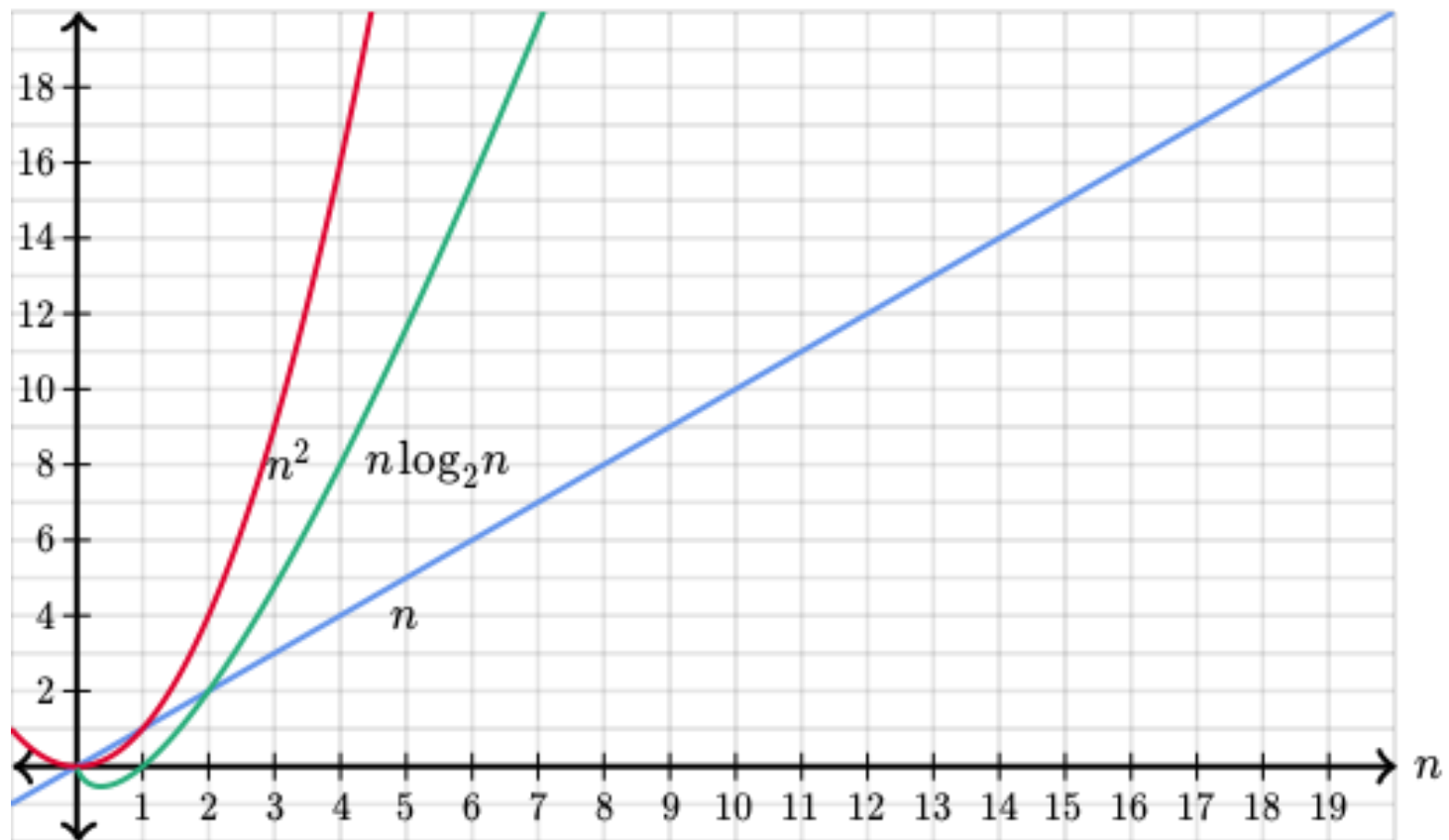
*Sorting and statistics.*

Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$ ✓	$\Theta(n \lg n)$ ✓
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)



# Sorting

*Sorting and statistics.*



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# Sorting - conclusions

- We introduced several algorithms that perform the sort in  $O(n \lg n)$  time.
- Merge sort and heap sort achieve this upper bound in the worst case.
- Quick sort achieves this limit in the average case.
- Furthermore, for each of these algorithms, one can obtain a sequence of  $n$  input data that makes the algorithm run in  $O(n \lg n)$  time.
- These algorithms share an interesting property: the determination of the order of the elements in their sorting is based only on comparisons between the input elements.

# Sorting - conclusions

- External sorting in **database** refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory, such as most data-base files.
- The typical external sorting algorithm uses a **sort-merge** strategy, which starts by sorting small subfiles—called runs—of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn.
- The **sort-merge** algorithm, like other **database algorithms**, requires buffer space in main memory, where the actual sorting and merging of the runs is performed.

# Questions and exercises:

1. What is sorting with linear time?
2. Explain how Merge Sort works ?
3. Explain how *Heap Sort* works ?
4. When is Quicksort better than Mergesort ?



# Thank you for your attention!

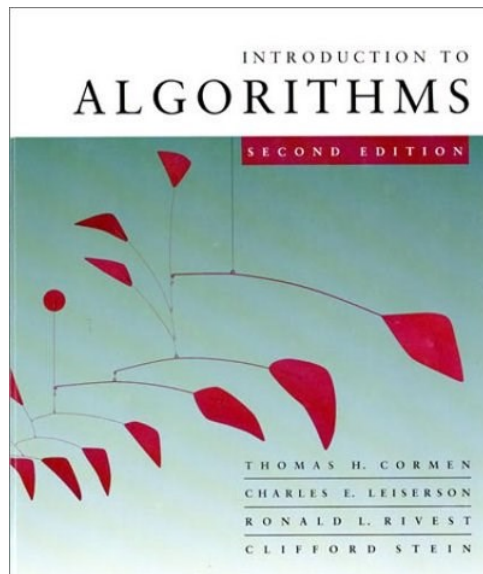
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 4. Graph algorithms**
  - ❑ Lesson 1. Introduction to graph algorithms.



ERASMUS+

# INTRODUCTION TO GRAPH ALGORITHMS



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT  
Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov  
(2002). Programming= ++Algorithms). Top Team  
Co, София.



# Introduction

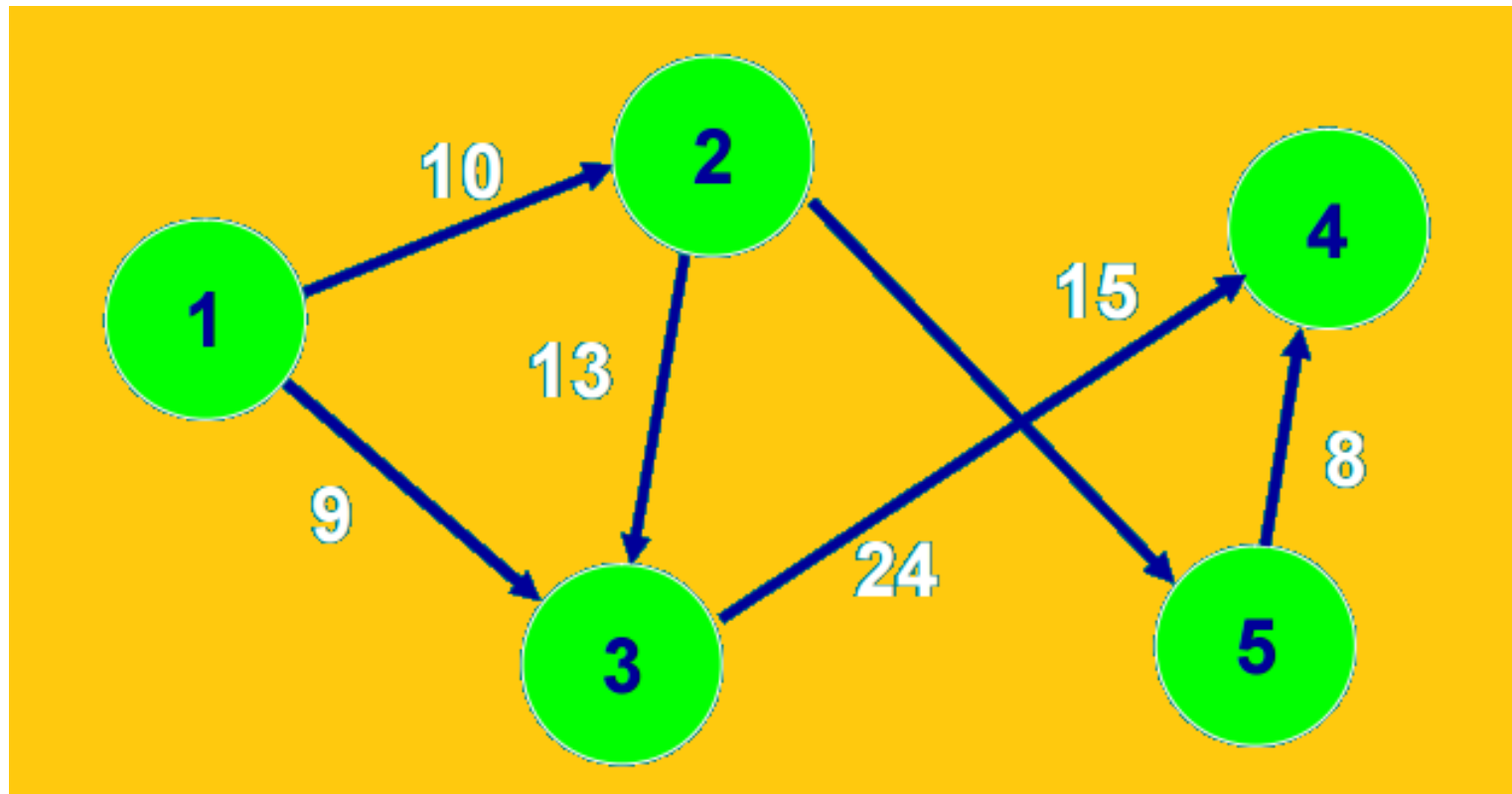
- This *lecture* presents methods for representing graphs and for searching graphs.
- Graph search techniques are the bases in the field of graph algorithms.
- A graph is viewed as a set of vertices (nodes) and arcs (edges).
- Presentation of a set of objects and their relationships is used.
- Usually the nodes correspond to the objects, the edges to the connections between them.

# Introduction

- Generally speaking, a graph is a data structure made up of vertices (nodes) and links between them, which are called edges.
- If the edges have a direction the graph is directed, otherwise it is undirected.
- Edges can also have a weight (or length). Then the graph is said to be **weighted graph**.
- Linked lists and trees are special cases of graphs.
- The following is an example of an directed, weighted graph:

# Introduction

- A directed, weighted graph:



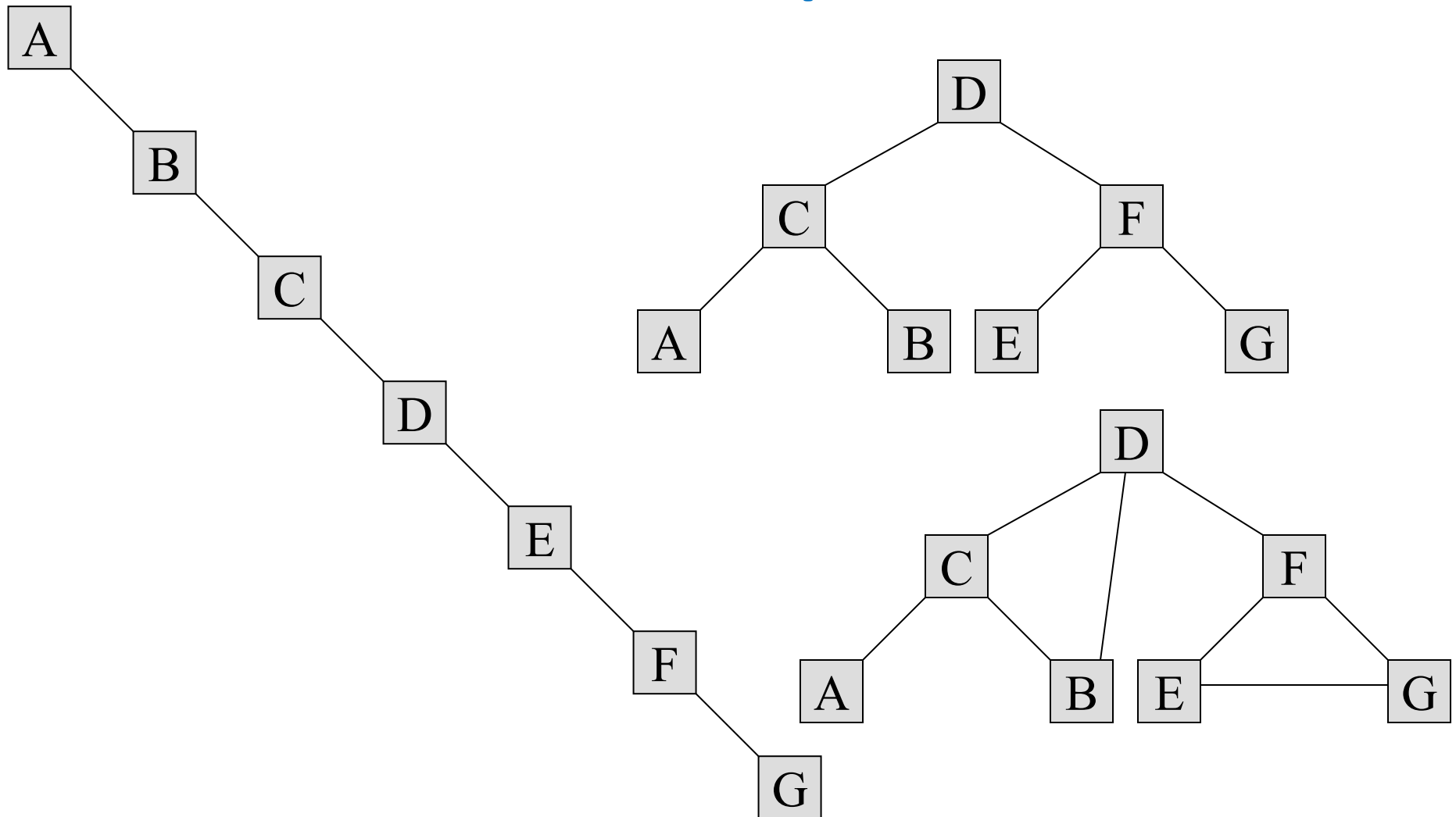
## And so! What Is A Graph?

- A collection of items, each of which can have zero or more successors and zero or more predecessors
- Trees and lists are just special cases of graphs

# Graphs in Everyday Life

- A road map
- A map of airline routes
- Links between Web pages
- Relationships in a social network
- Diagram of flow capacities in a communication or transportation network

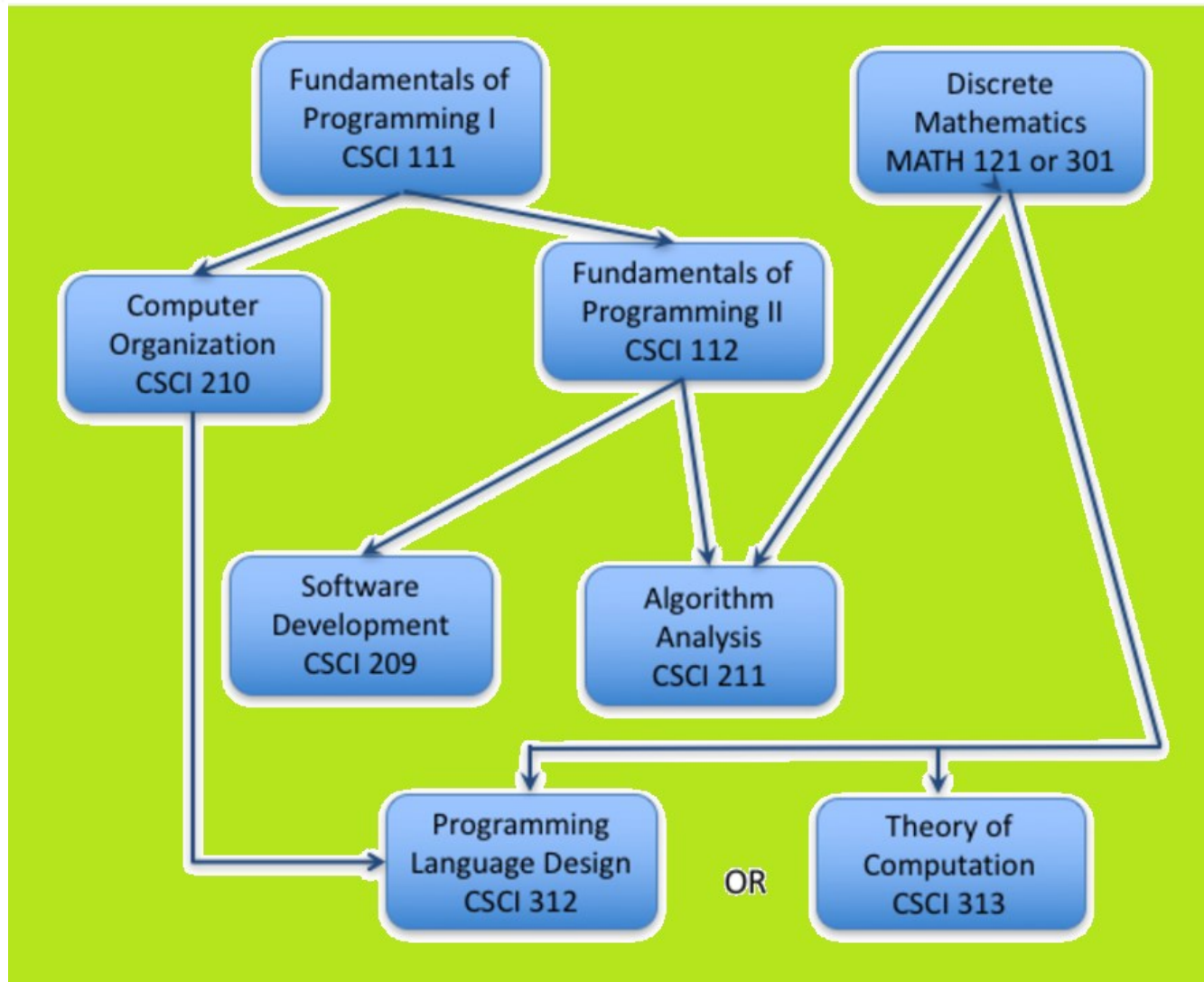
# Examples



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

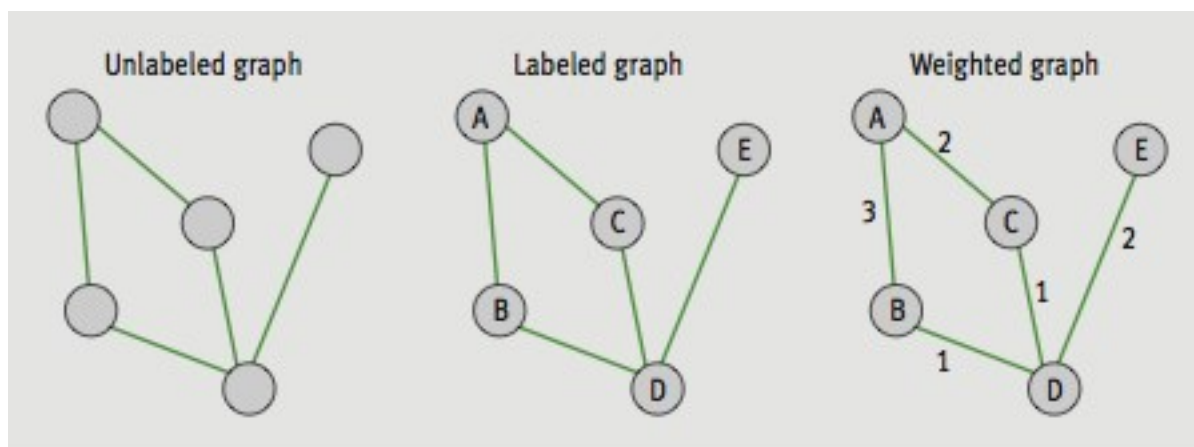




# Vertex, Edge, Label, and Weight

- The nodes in a graph are also called *vertices*
- The connections between vertices are called *edges*
- Vertices and edges can be labeled or unlabeled
- A numeric edge label is also called a *weight*

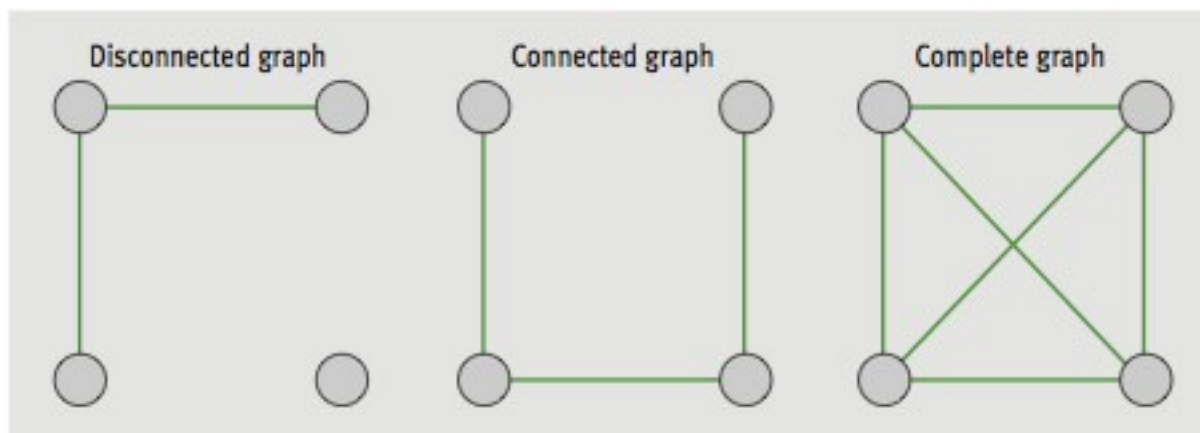
# Examples



## Connections

- A graph is *connected* if there is at least one edge from each vertex to some other vertex
- A graph is *complete* if there is an edge from each vertex to every other vertex

# Examples

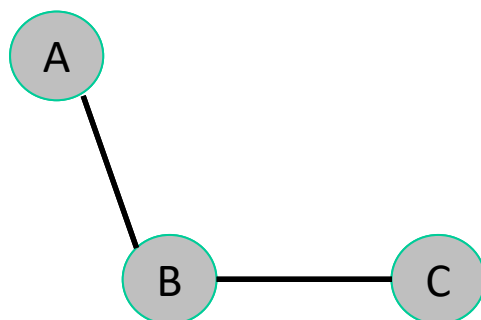


## Paths and Cycles

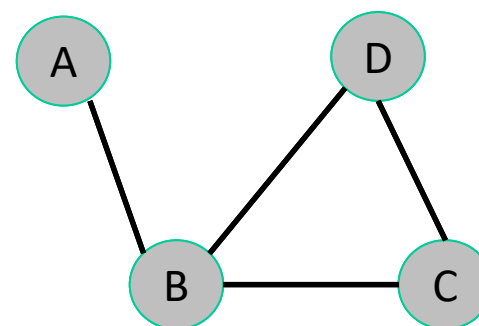
- A *path* is a sequence of edges that allows one vertex to be reached from another vertex
- A *simple path* is a path in which a vertex is not visited more than once
- A *cycle* is a path in which a vertex is visited more than one

# Examples

Simple path: ABC

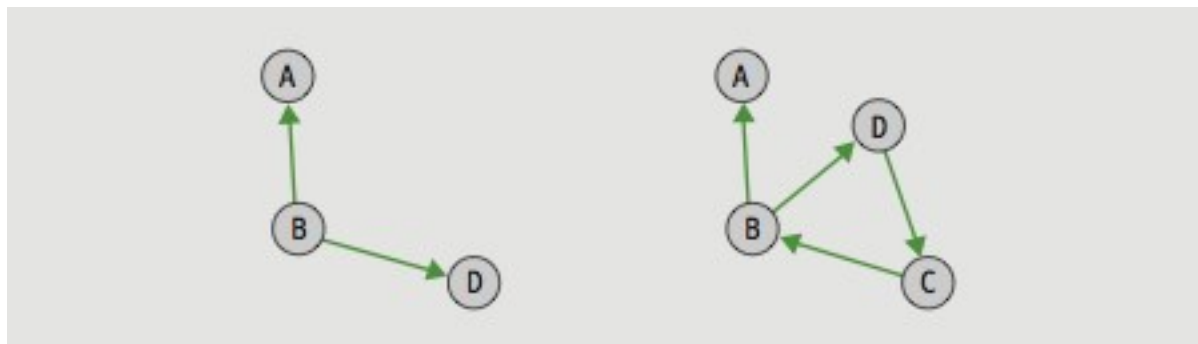


Cycle: BCD



# Directed Graphs (Digraphs)

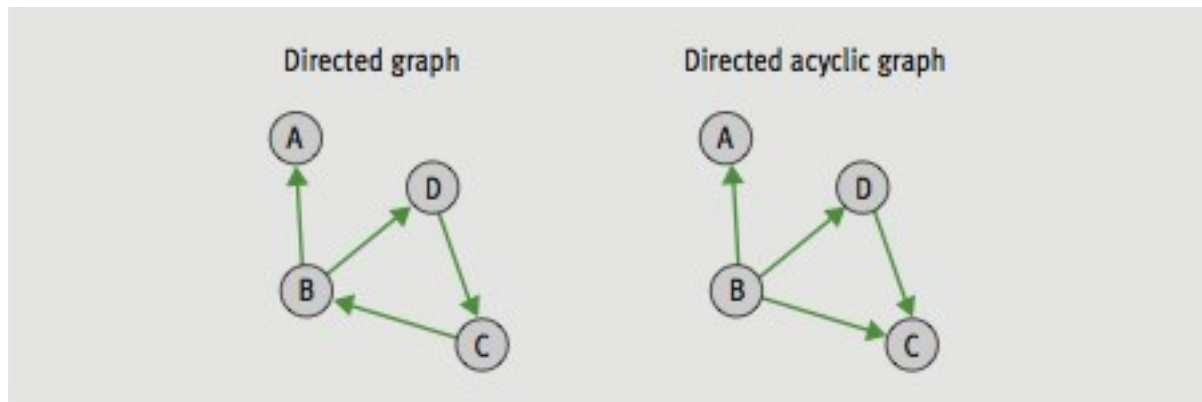
- Each edge in a *directed graph* points in one direction, allowing movement from a source vertex to a destination vertex
- These edges are called *directed edges*





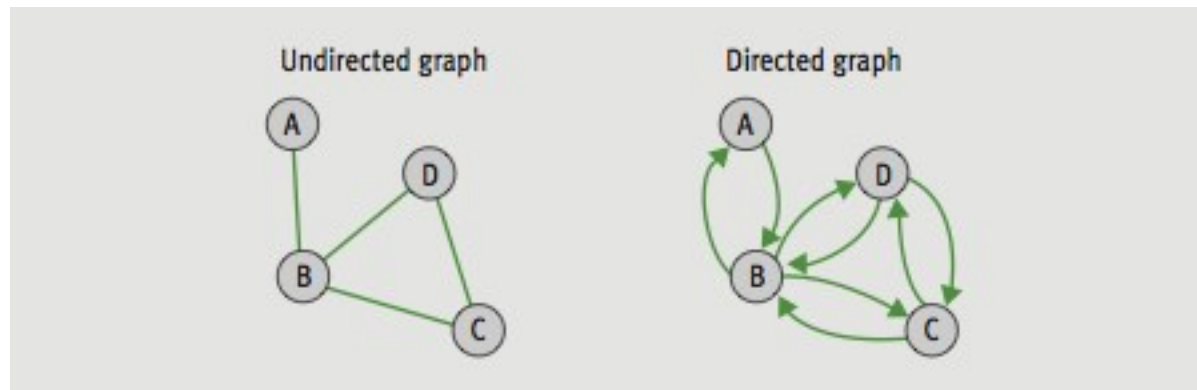
# Directed Acyclic Graphs (DAGs)

- A *directed acyclic graph* is a directed graph with no cycles



# Directed and Undirected Graphs

- The edges in an *undirected graph* support movement in both directions

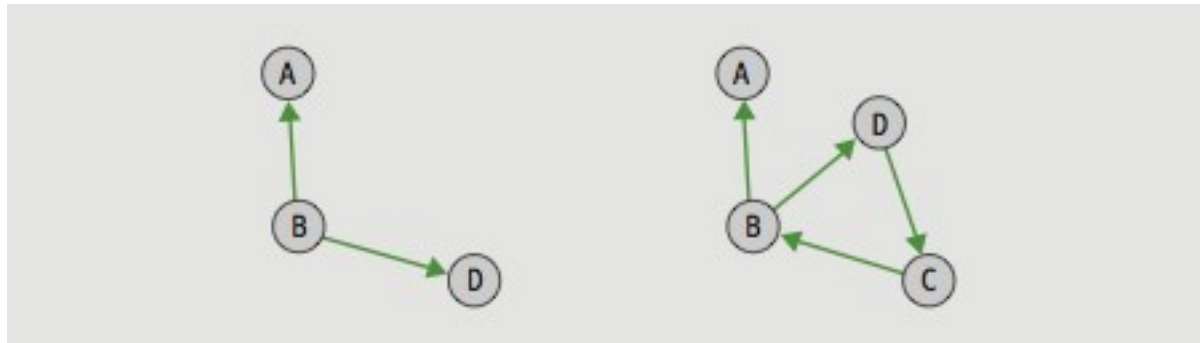


# Sparse and Unsparse Graphs

- *Sparse graphs* are connected graphs with a minimal number of edges (roughly  $N$  edges)
- *Unsparse graphs* have close to the maximum number of edges (roughly  $N^2$  edges)

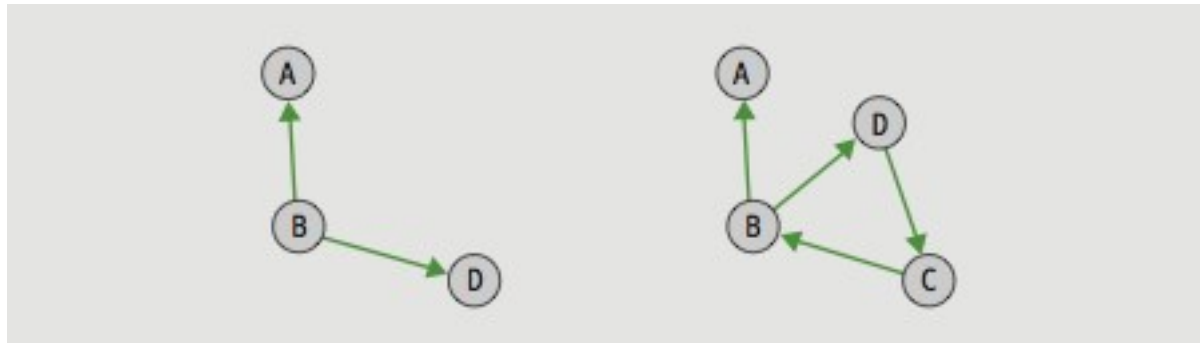
# Adjacent Vertices

- Vertex  $A$  is *adjacent* to vertex  $B$  if there is a directed edge from  $B$  to  $A$



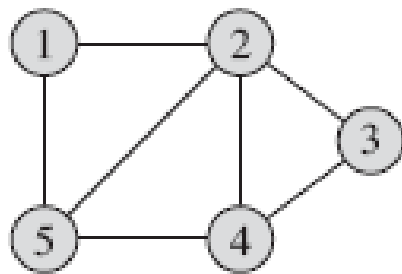
# Incident Edges

- Edge BA is *incident* to vertex B if it is a directed edge from B to A



## Presentation of graphs

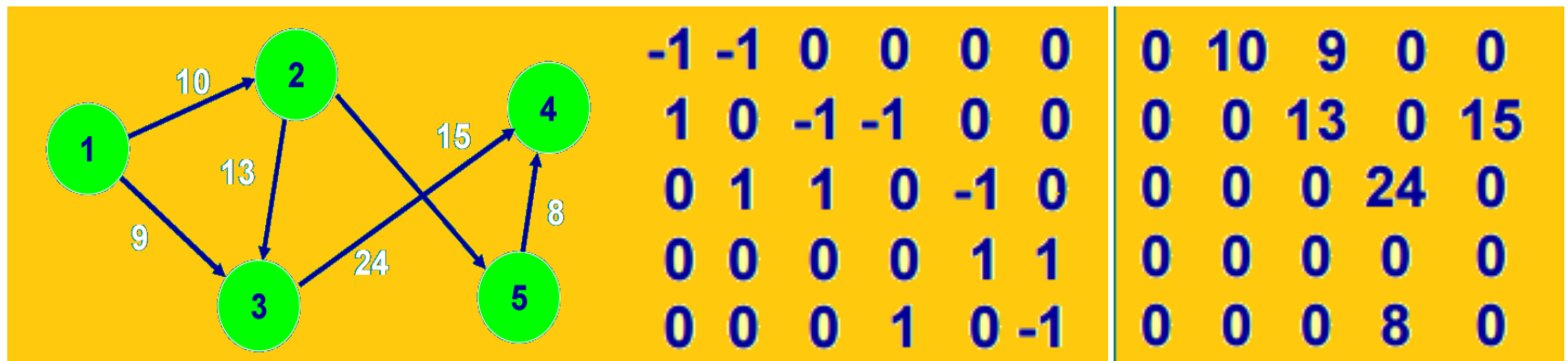
- **Presentation of graphs:** One possible way to represent a graph is by a neighborhood matrix. It consists of  $n$  rows and  $n$  nodes, where  $n$  is the number of nodes in the graph. Each row and column corresponds to a specific node. If there is an edge between node 1 and node 2 then the element of position  $[1][2]$  is 1, and if there is no edge - 0.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

# Presentation of graphs

- **Presentation of graphs:** Another possible way to present a graph is by the incidence matrix. In it, the rows represent the nodes and the pillars represent the edges. The element of position  $[ i ][ j ]$  is -1 if edge  $j$  comes out of node  $i$ , 1 if edge  $j$  comes in node  $i$ , and 0 otherwise.



*When the graph is weighted, we can record the weights of the edges instead of units in the neighborhood matrix.*

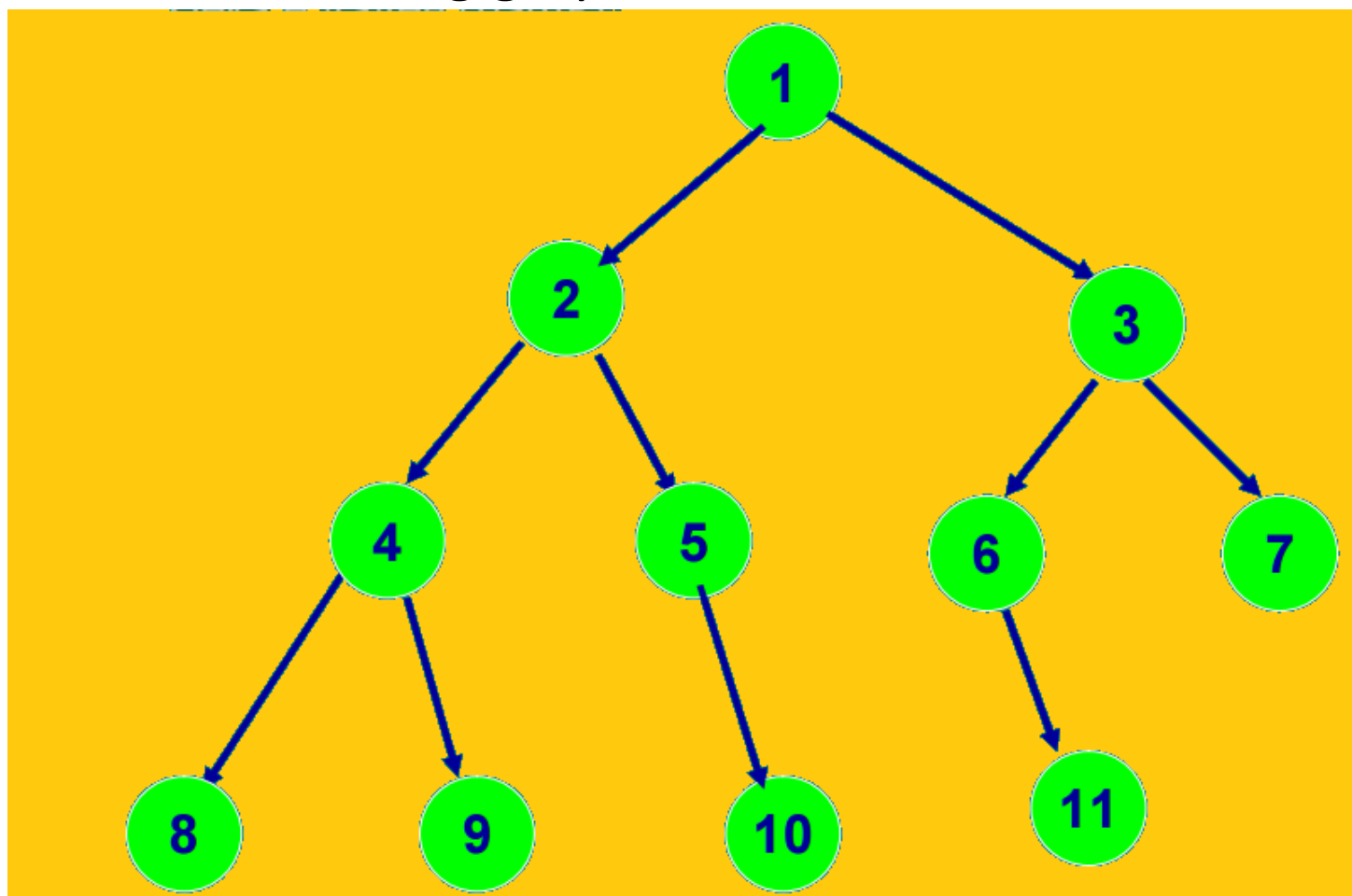
# Graph traversal

- Two graph search algorithms can also be implemented: in depth and in breadth.
- In **depth-first search**, one starts from some initial node and moves to its successor, then to the successor of the successor, and so on until one reaches a node with no successors (from which no edges go out). Then a **return** is made to the previous node.
- In **breadth-first search**, one starts from some initial node and searches all its successors, only then searches the successors of the successors.



# Graph traversal

- Given the following graph:



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# Graph traversal

- If we traverse the successors of a node in increasing order of their numbers, then:

- in **depth-first search**, the nodes will be traversed in the following sequence:

1 2 4 8 9 5 10 3 6 11 7

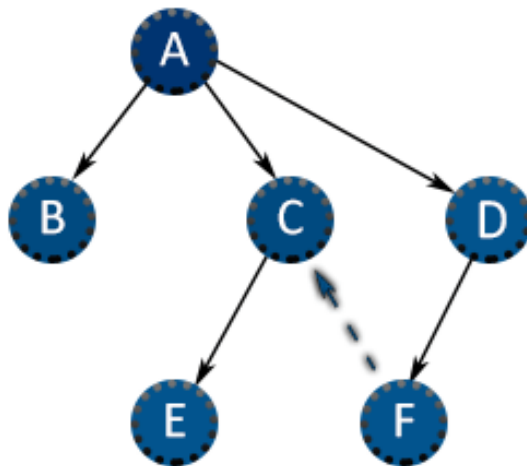
- in **Breadth-first search**, the nodes will be traversed in the following sequence:

1 2 3 4 5 6 7 8 9 10 11

# Graph traversal

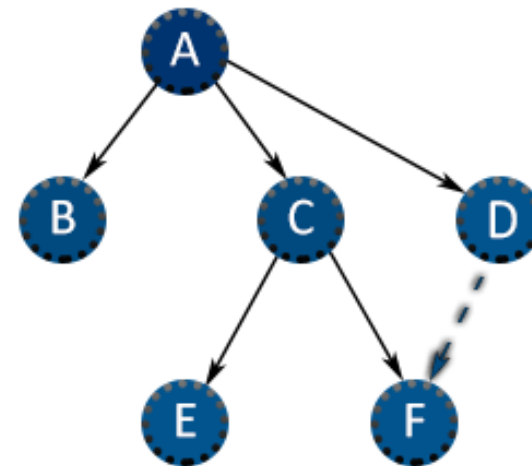
- Given the following graph:

**DFS**



**A D F C E B**

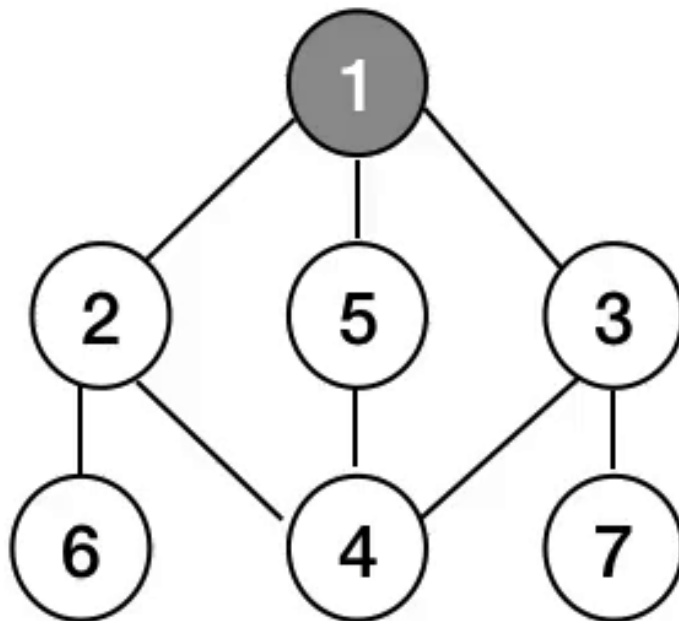
**BFS**



**A B C D E F**

# Graph traversal

- Given the following graph:



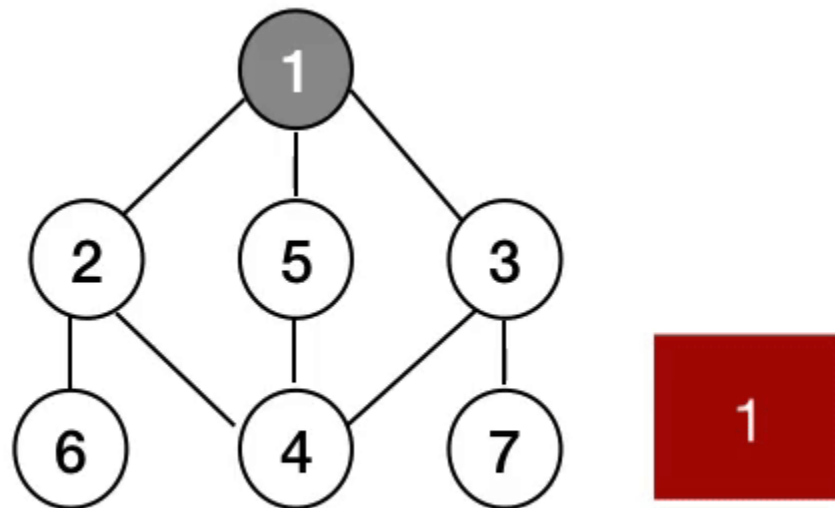
**DFS**

1

# Graph traversal

- Given the following graph:

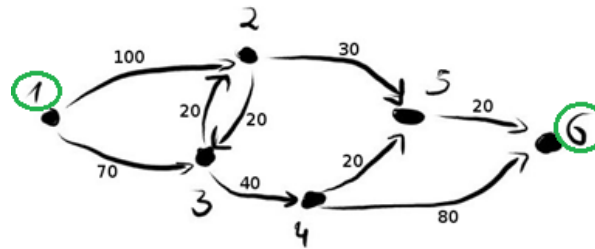
**BFS**



# Graph traversal

## application in DB

- Let's take a real-life example. Let's assume we've got a **database** with a list of nodes and a list of links between them (you can think of them as cities and roads). Our task is to find the shortest path from node 1 to node 6.



- Well, in fact, it's nothing more than graph traversal. The very first idea we may have would be to get all rows from both tables and implement a **DFS** (Depth-First Search) or **BFS** (Breadth-First Search) algorithm with a single **SQL query**!

## Questions and exercises:

1. What are different methods for representing graphs ?
2. What are different methods for searching in graphs ?
3. What is a graph and what is it used for?



# Thank you for your attention!



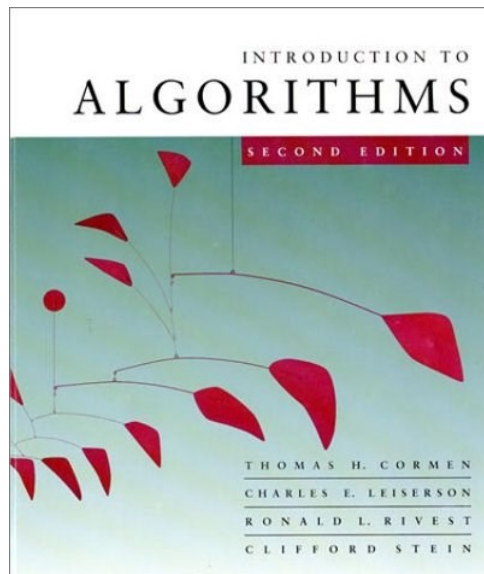
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 4. Graph algorithms**
  - ❑ Lesson 2. Tree Cover of Graphs.



ERASMUS+

# TREE COVER OF GRAPHS

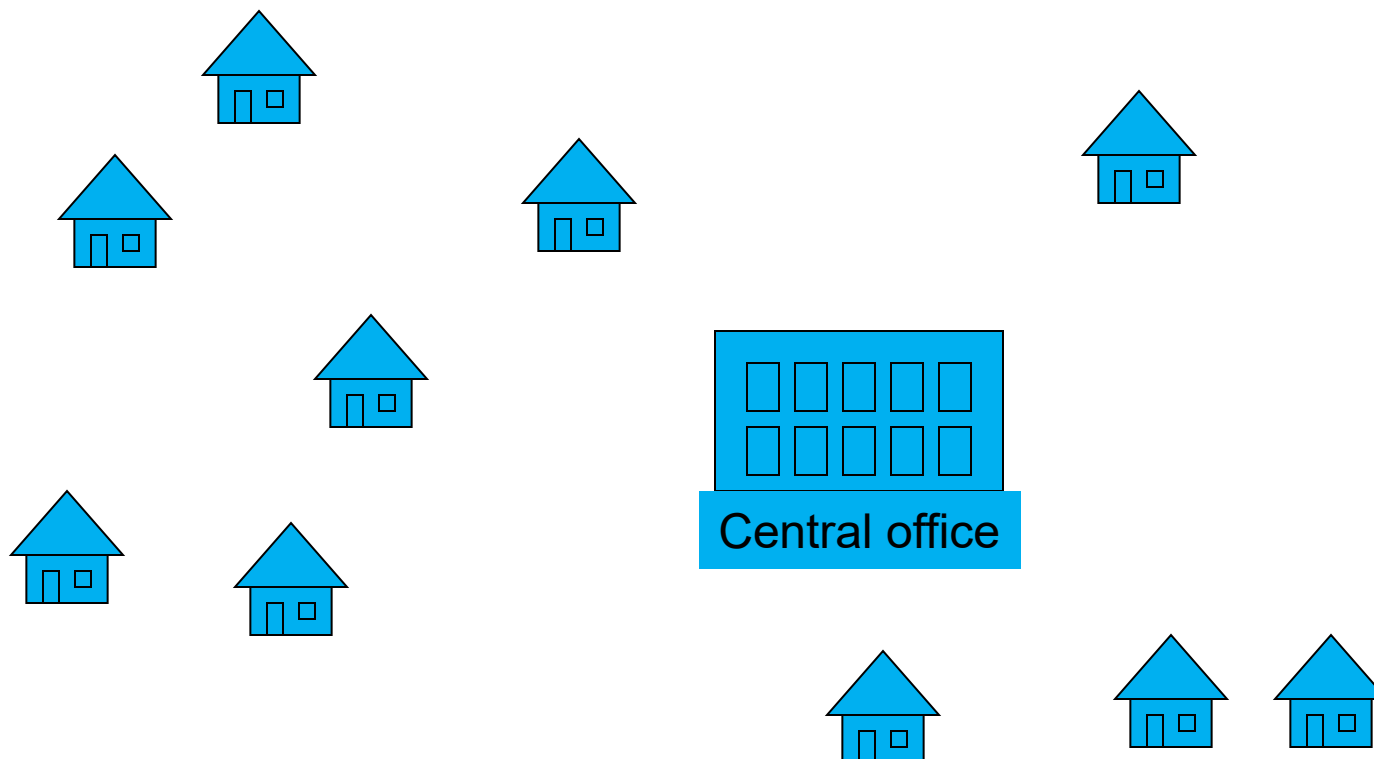


[Carmen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT  
Press and McGraw-Hill.

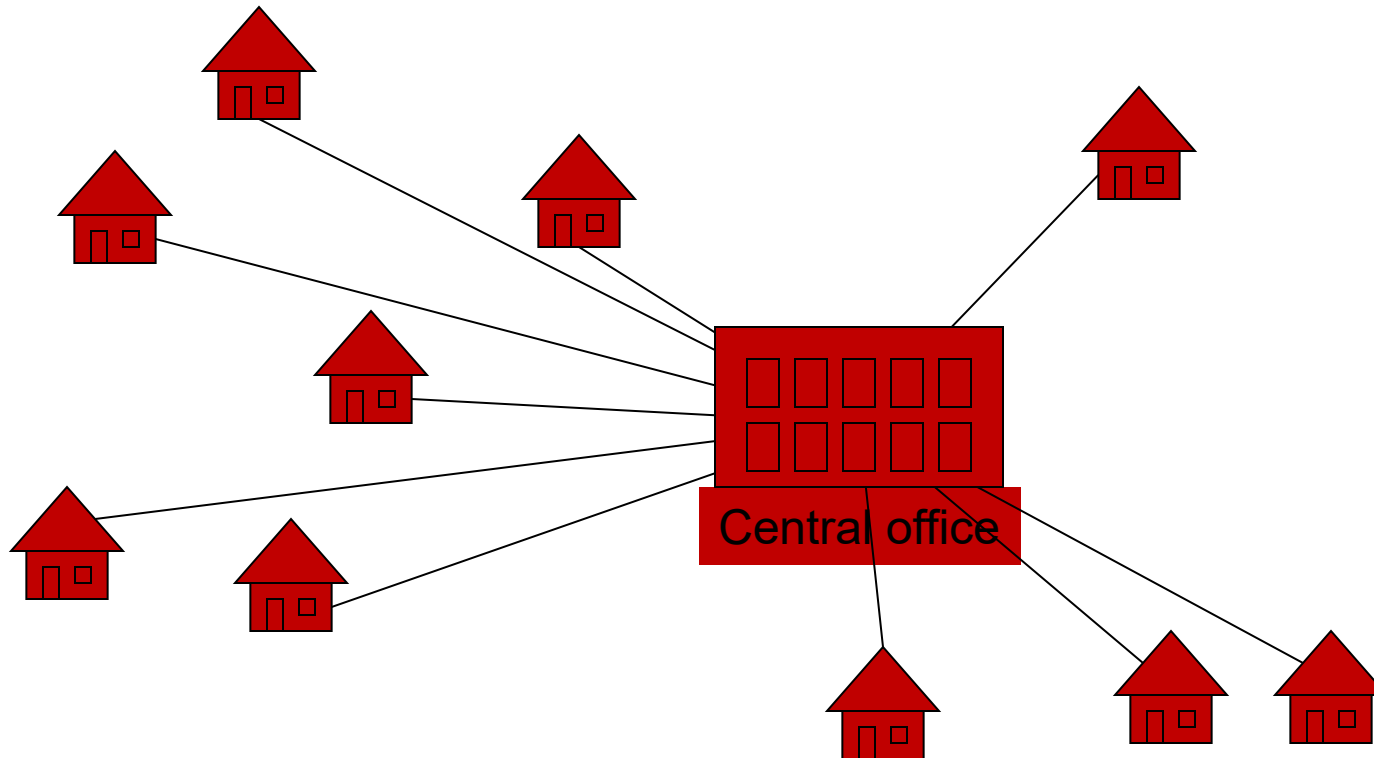


Preslav Nakov, Panayot Dobrikov  
(2002). Programming= ++Algorithms). Top Team  
Co, София.

# Problem: Laying Telephone Wire

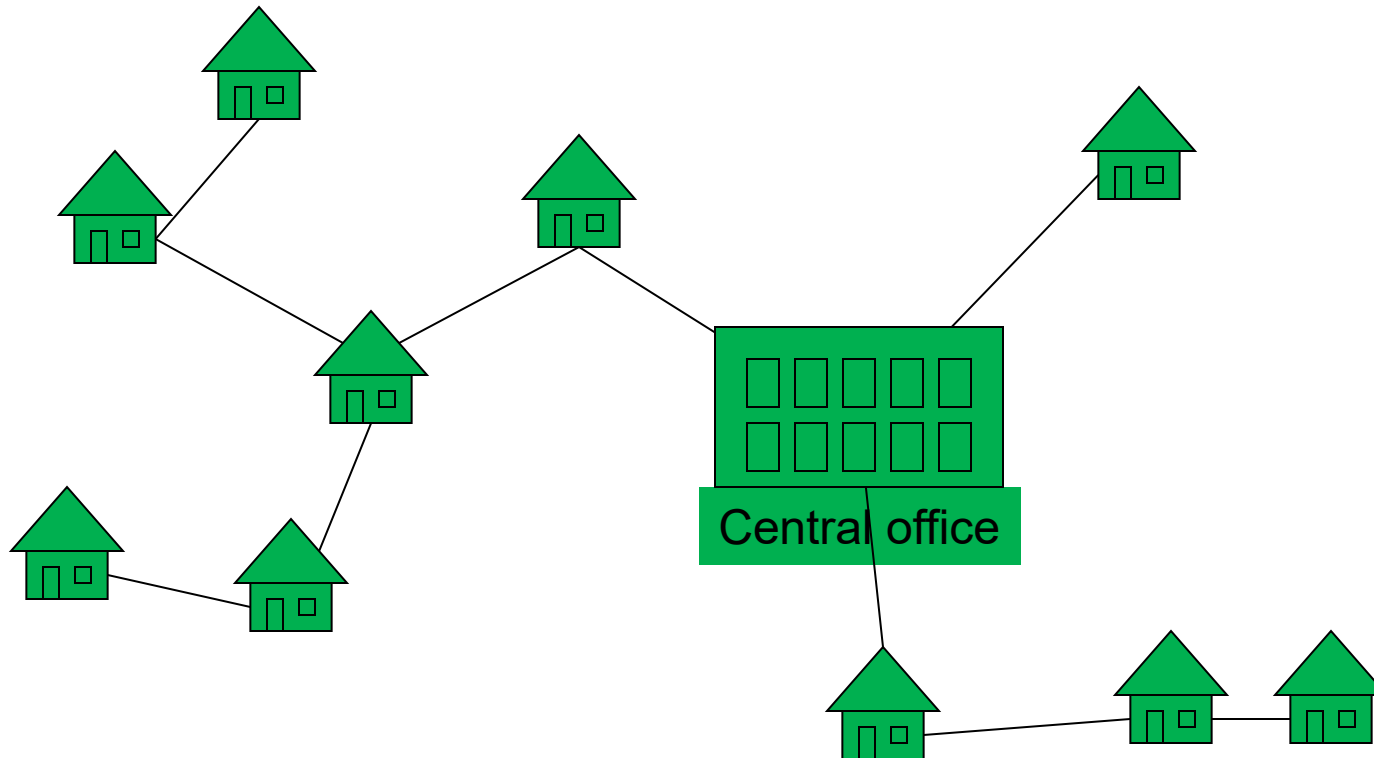


# Wiring: Naive Approach



**Expensive!**

# Wiring: Better Approach



Minimize the total length of wire connecting the customers.

## Cover trees

- Let  $G = (V, A)$  be a random graph. Any tree formed by edges of  $G$  including all nodes of the graph is called a ***cover (spanning) tree***.
- Many real tasks in economics, management, etc. can be interpreted as problems of finding (searching, constructing) cover trees satisfying some optimality conditions.
- In many problems it is necessary to find not just some cover tree, but a tree with optimal properties. In the undirected graph  $G = (V, A)$ , let each edge  $(u, v)$  have weight  $c(u, v)$ , where the weight of the tree is the sum of the weights of the participating arcs.

## Cover trees

- **Example:** *A construction company is looking at a project to build roads between six settlements to provide communications between these settlements. The costs (prices) required to lay each of the possible routes between these settlements are known. The company wants to provide the road communications at minimum cost.*
- Consider the undirected graph  $G = (V, A)$  whose **nodes** correspond to the **cities**, the **edges** to the **roads** that can be routed between these cities, and the weights of the edges are the corresponding costs of routing the road. Clearly, in this case the company's strategy comes down to finding a cover tree for this graph with minimum weight (cost).

## Cover trees

- The edges of the output graph are considered in random order, and at each step of the algorithm a decision is made whether or not to include the corresponding edge in the cover tree.
- In this case, the edge that is included in the tree is colored **green** and the one that is not included is colored **black** i. e. the algorithm is a process of coloring the edges. The colored edge is not considered further.
- At each step in the algorithm, a check is made whether the edge under consideration in the set with the **green** (i. e. already included in the tree) edges forms a loop. If so - the edge is colored **black** (i. e. the cover tree is not included).



## Cover trees

- **The green** (included in the tree) edges form one or more components.
- The nodes of each component form a set of nodes, which we will call a bunch.
- Therefore, the edge under consideration will form a loop with the edges included in the tree if both of its nodes belong to one of the bunches formed so far.
- In other words, in addition to coloring the edges, the algorithm maintains (stores) and updates the nodes bunches of the individual connected components.

## Cover trees

- Description of the algorithm for finding a cover tree:
- STEP 1. We choose a random edge. We color this edge **green**. We form a bunch from the nodes of this edge.
- STEP 2. We choose a random uncolored edge. If there is no such edge, proceed to step 4.
- STEP 3. The following four cases are possible:

## Cover trees

- Description of the algorithm for finding a cover tree:
  - a) none of the nodes of the edge belongs to a previously formed bunch - color the edge **green** and form a new bunch from the nodes of this edge. Move on to step 4.
  - б) both edges belong to the same vertex bunch formed so far - we color the edge **black**, i. e. we do not include it in the tree we are building. Move on to step 4.

## Cover trees

- Description of the algorithm for finding a cover tree:
  - b) one vertex of the edge belongs to a given bunch and the other does not belong to any of the bunches formed so far - color the edge **green** and include the vertex not included in the bunch containing the other vertex. Move on to step 4.
  - r) the edge nodes belong to different bunches - color the edge **green** and merge the two bunches into one new bunch. Move on to step 4.

## Cover trees

- Description of the algorithm for finding a cover tree:
- STEP 4. If all vertices of the graph are in a bunch (the number of colored edges is one less than the number of vertices of the graph), the edges colored green form a cover tree for the graph. End.
- *STEP 5. Move on to step 2.*

*Such formulated algorithm "works" well when a cover tree exists and "loop" if none exists.*

## Cover trees

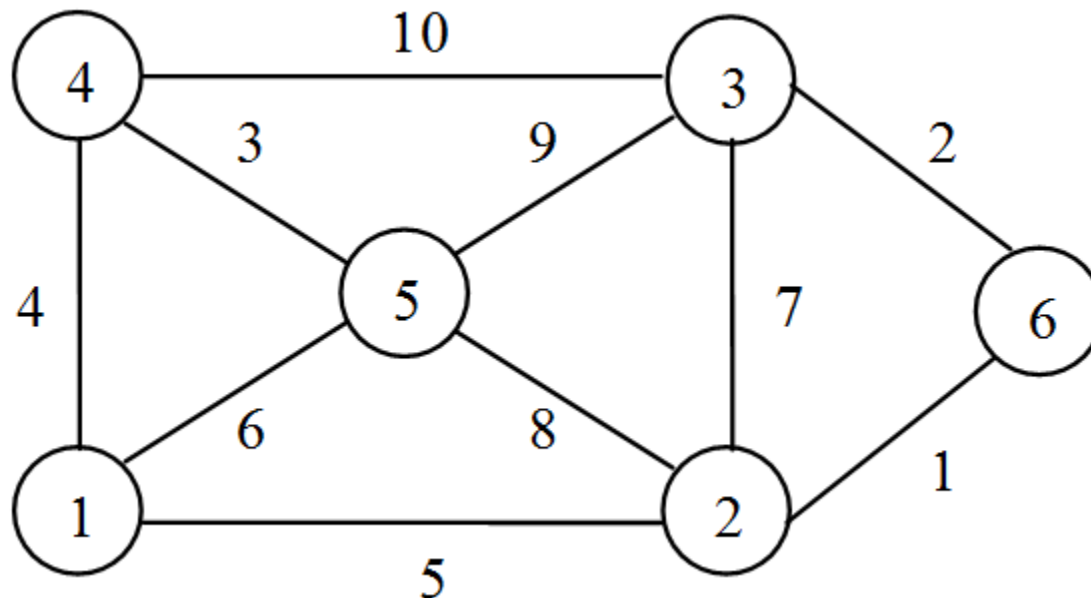
- Description of the algorithm for finding a cover tree:

Solution proposal by removing the "looping": In step 4, we check if all edges of the graph are colored. If this is the case, and the number of edges colored green is less than the number of vertices of the graph minus one, then there is no covering tree for the graph.

## Cover trees

- I will illustrate the algorithm with the following example:

Example 2: Given a graph  $G = (V, A)$



Construct a cover tree for this graph.

## Cover trees

- Solution:

14	Edge	Color	Bunch 1 (of nodes)	Bunch 2 (of nodes)	Bunch 3 (of nodes)	...
1	(1, 5)	Green	1, 5	$\emptyset$	$\emptyset$	
2	(2, 6)	Green	1, 5	2, 6	$\emptyset$	
3	(4, 3)	Green	1, 5	2, 6	4, 3	
4	(5, 3)	Green	1, 5, 4, 3	2, 6	$\emptyset$	
5	(1, 4)	Black	1, 5, 4, 3	2, 6	$\emptyset$	
6	(5, 2)	Green	1, 5, 4, 3, 2, 6	$\emptyset$	$\emptyset$	

Since after considering and coloring the sixth edge, all the vertices of the graph turn out to be in one bunch (or which is the same - the number of edges colored green is one less than the number of vertices of the graph), the tree consisting of the edges (1, 5), (2, 6), (4, 3), (5, 3), (5, 2) is the covering tree for the initial graph.



## Cover trees

- Solution:
- The weight of the found covering tree is:  
 $c(1, 5) + c(2, 6) + c(4, 3) + c(5, 3) + c(5, 2) = 6 + 1 + 10 + 9 + 8 = 34.$
- Now let's apply the algorithm to the same graph, considering its edges in the order of their increasing weights.

## Cover trees

- Solution:

No	Edge	Color	Bunch 1 (of nodes)	Bunch 2 (of nodes)	....
1	(6, 2)	green	6, 2	$\emptyset$	
2	(6, 3)	green	6, 2, 3	$\emptyset$	
3	(4, 5)	green	6, 2, 3	4, 5	
4	(4, 1)	green	6, 2, 3	4, 5, 1	
5	(1, 2)	green	6, 2, 3, 4, 5, 1	$\emptyset$	

The weight of the obtained cover tree in this case is:

$$c(6, 2) + c(6, 3) + c(4, 5) + c(4, 1) + c(1, 2) = 1 + 2 + 3 + 4 + 5 = 15 \text{ and is the maximum cover tree.}$$

# Cover trees

- We can formulate the following two statements:
- St. 1. Minimum cover tree search algorithm.

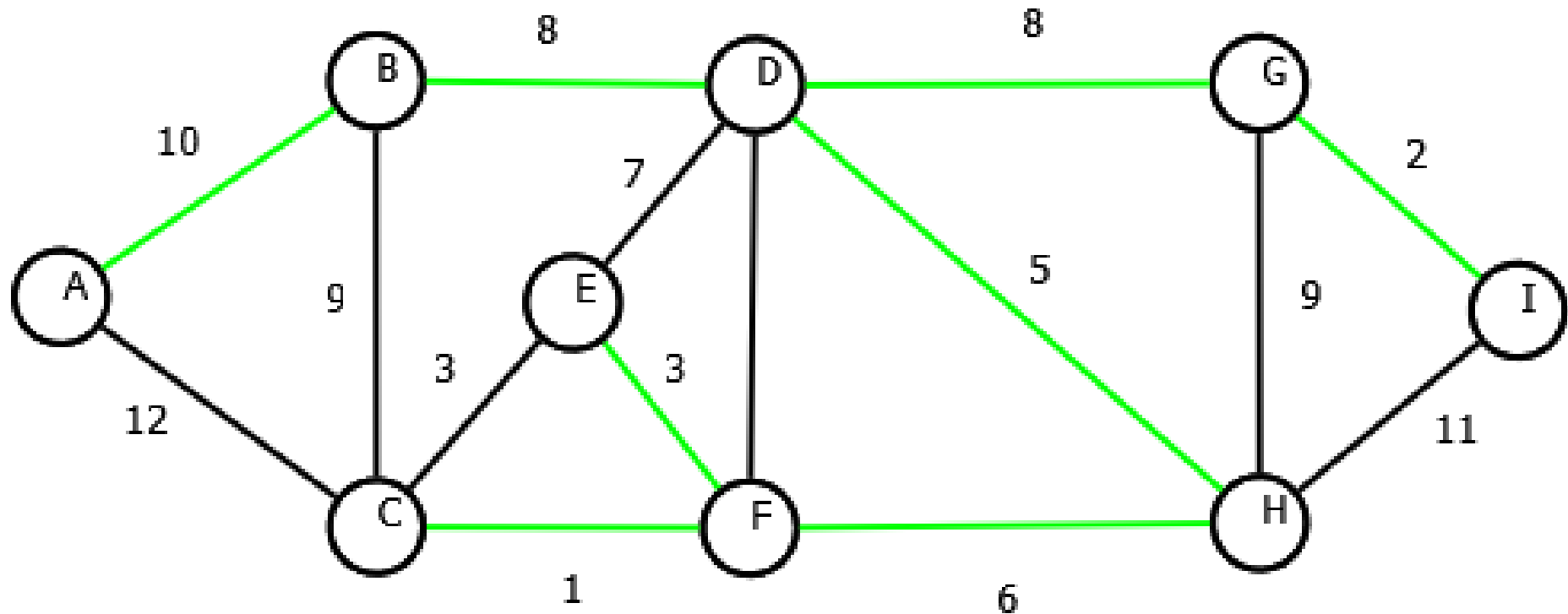
The cover tree search algorithm is applied, with edges considered in order of increasing their weights. If there are edges with equal weights, they are considered in random order.

- St. 2. Maximum cover tree search algorithm.

The covering tree search algorithm is applied, with edges considered in order of decreasing their weights. If there are edges with equal weights, they are considered in random order.

# Cover trees

- Minimum Spanning Trees example

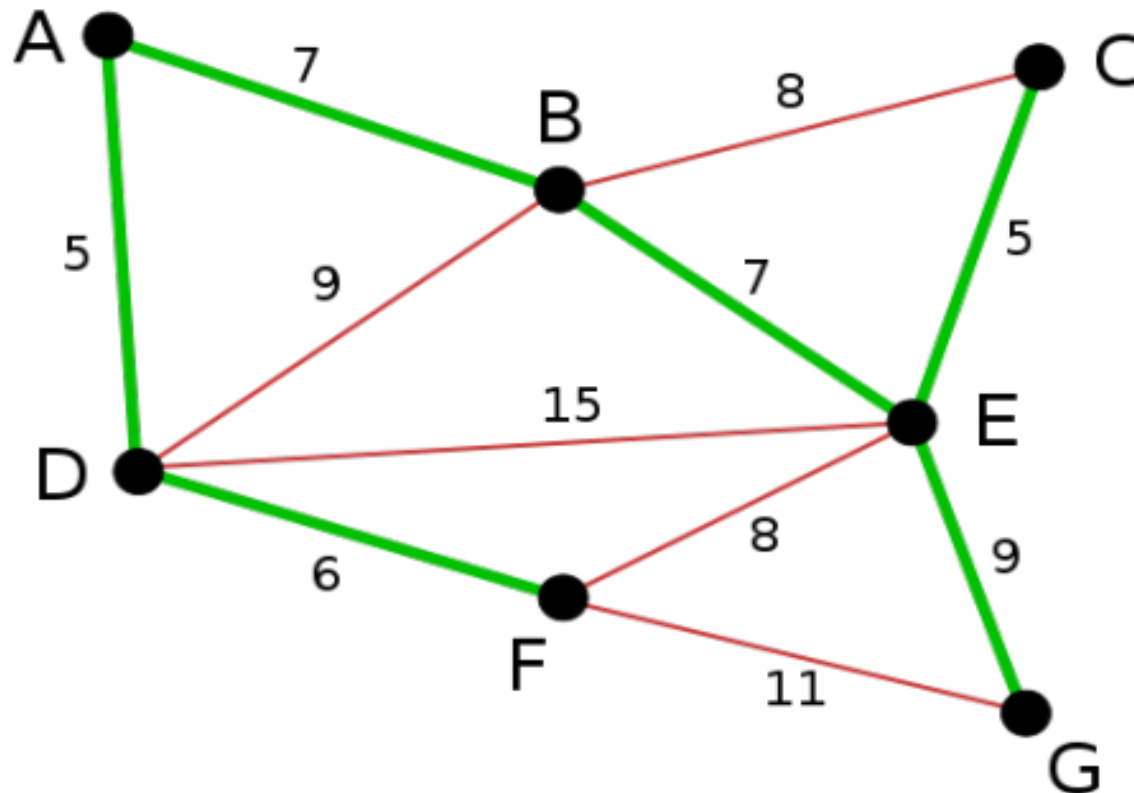


# Cover trees

- **Kruskal's Minimum Spanning Tree Algorithm**
- Kruskal has proposed a **greedy algorithm**. Let our tree be  $G^T = (V, T)$ , on the graph  $G = (V, E)$ :
  1. We divide the  $N$  vertices of the graph into  $N$  separate sets  $S_i, i = 1 \dots N$ ;
  2. We sort the edges by decreasing value of their weights;
  3.  $N - 1$  times we choose the edge  $u$  with the least weight for which  $u \notin T$  and which connects two separate sets  $S_i$  and  $S_j$  into a new set  $S_k = S_i \cup S_j$ ;

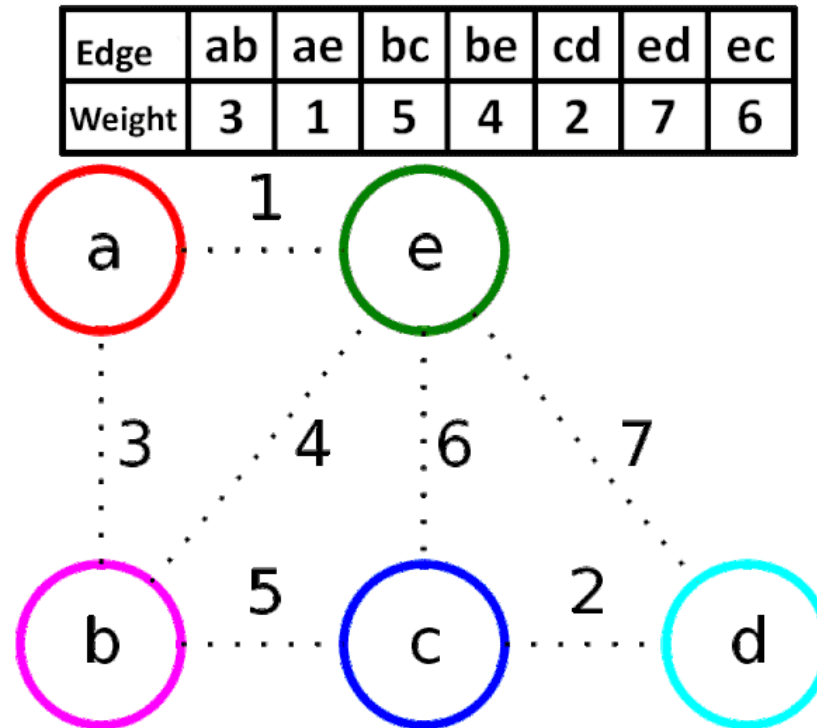
# Cover trees

- Minimum Spanning Tree: Kruskal's Algorithm



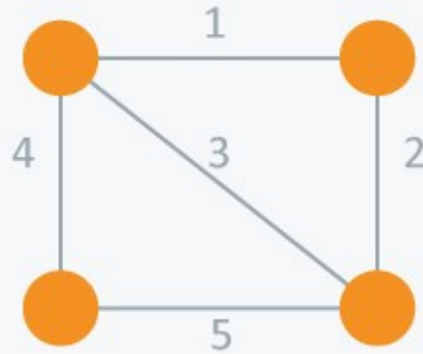
# Cover trees

- Minimum Spanning Tree: Interactive visualisation of Kruskal's Algorithm

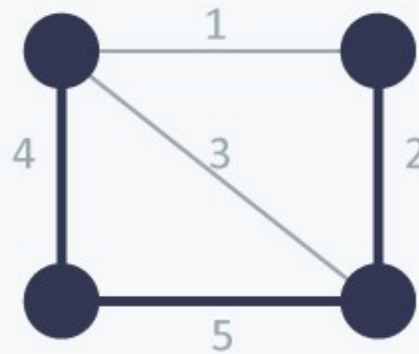


ERASMUS+

# Cover trees

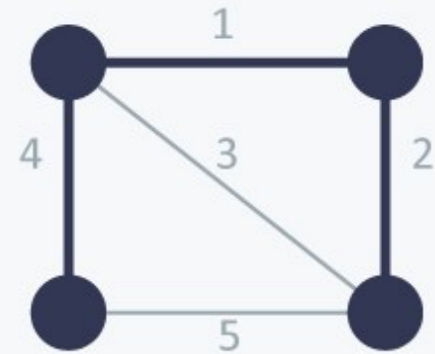


Undirected  
Graph



Spanning  
Tree

Cost =  $11(=4+5+2)$



Minimum Spanning  
Tree

Cost =  $7(=4+1+2)$

**In a database, Kruskal's algorithm can be used in the query optimization process for the generation of the execution query tree.**



## Questions and exercises:

1. What does cover of graphs problem involve ?
2. What is a minimum spanning tree ?
3. How do you find the weight of an edge in a minimum spanning tree?
4. What are some algorithms used to create a minimum spanning tree?
5. What are the disadvantages of using Kruskal's algorithm on large graphs with many edges?



# Thank you for your attention!

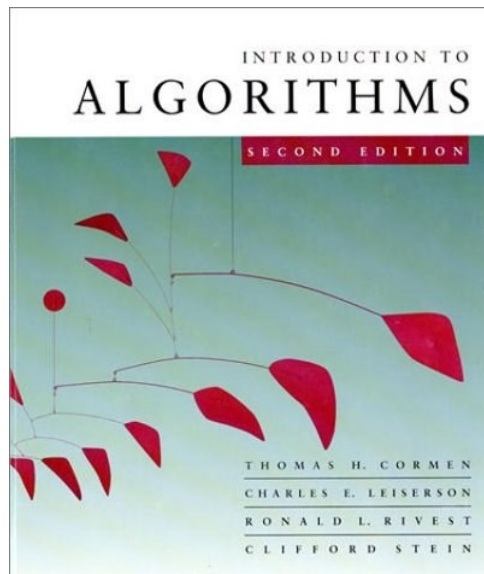
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 4. Graph algorithms**
  - ❑ Lesson 3. Shortest Path Algorithms.



ERASMUS+

# SHORTEST PATHS ALGORITHMS



[Cormen, Thomas H.](#); [Leiserson, Charles E.](#), [Rivest, Ronald L.](#), [Stein, Clifford](#) (2009)  
[1990]. Introduction to Algorithms (3rd ed.). MIT  
Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov  
(2002). Програмиране = ++Алгоритми). Top Team  
Со, София.

# Introduction

- A shortest path in a graph between two vertices is called a path that starts at one vertex (node) and ends at the other, and the sum of the weights of the edges involved is minimal.
- **Definition.** Let be given a weighted directed graph  $G(V, E)$  with edge weights given real numbers. *The length of a path in  $G$*  is called the *sum* of the weights of the edges in it.
- In some problems, it is possible to define the path length not as a *sum*, but as some other *function* of the weights of the edges (and even vertices) involved in the path.

# Shortest paths in graphs

- In order for these algorithms to remain valid in these cases, some specific optimality criteria must be satisfied (they depend on the particular algorithm under consideration).
- Since there may be no restriction that the path be simple, we should be careful in cases where the graph contains a loop.
- So, for example, if we are looking for a minimum path, and there is a negative cycle (a loop of negative length), we will be able to "pivot" on that cycle random number of times, where the length of the path (equal to the sum of the edges in it) will decrease arbitrarily much toward minus infinity.

## Shortest paths in graphs

- Similarly, if we are looking for a maximal path and there is a positive cycle, then for each path that contains it, we can "pivot" on the cycle, obtaining an arbitrarily large length.
- *Can the shortest path contain a cycle?* As we have just seen, it cannot contain a single negative weight cycle.
- It also cannot contain a positive-weight cycle, since removing that cycle from the path produces a path with the same source and vertices destination and a lower path weight.

# Shortest paths in graphs

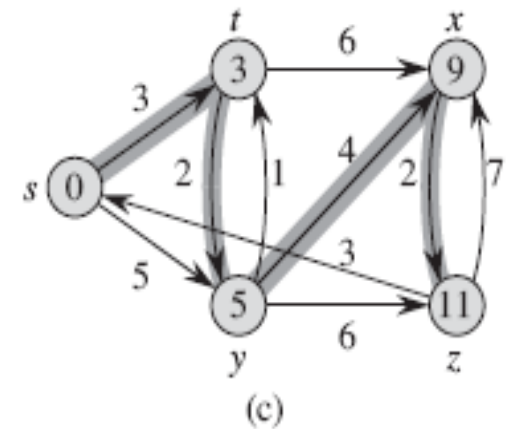
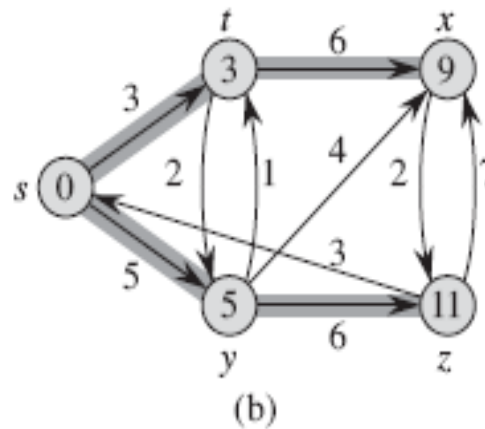
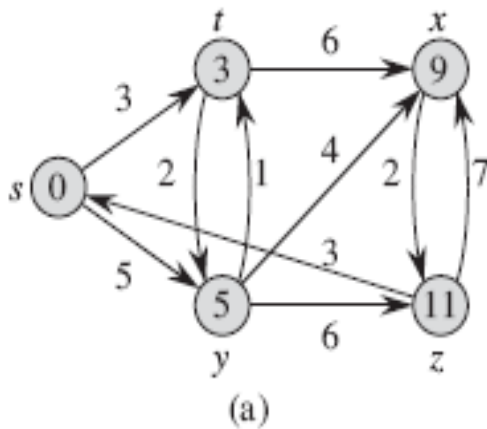
- So if  $p = \langle v_0, v_1, \dots, v_k \rangle$  is a path and  $c = \langle v_i, v_{i+1}, \dots, v_j \rangle$  is a cycle with a positive weight on that path (so that  $v_i = v_j \wedge w(c) > 0$ ), then the path  $p' = \langle v_0, v_1, \dots, v_i, v_{j+1}, v_{j+2}, \dots, v_k \rangle$  has weight  $w(p') = w(p) - w(c) < w(p)$ , and so  $p$  cannot be the shortest path from  $v_0$  to  $v_k$ .
- So that only *cycles with weight 0* are left. We can remove cycles with weight 0 from each path and produce a path with the same weight.
- Thus, if there is a shortest path from a vertex  $s$  (source) to a destination  $v$  vertex containing a cycle with weight 0, then there is another shortest path from  $s$  to  $v$  in this graph without this cycle.



## Shortest paths in graphs

- As we search for a shortest path in a graph with multiple cycles of weight 0, we may repeatedly remove such detected cycles until we find a shortest path in the graph without these cycles.
- **Therefore**, we can assume that when we find shortest paths, they do not have the cycle, i. e., they are simple paths.
- Since every acyclic path in a graph  $G=(V, E)$  contains at most  $|V|$  different vertices, contains the most  $|V|-1$  arcs (edge).
- We can focus on finding the shortest path with the most  $|V|-1$  arcs.

# Shortest paths in graphs



- a) Weighted directed graph with shortest path weights from source  $s$
- b) The colored arcs form a tree of shortest paths from source  $s$ .
- c) Another tree on a short path with the same root  $s$ .

## Dijkstra's algorithm

- The most efficient method for finding the minimum paths from one particular vertex to all others is the Dijkstra algorithm. **It is a type of greedy algorithm.** It only works on weighted graphs with positive weights.
- Let be given a weighted directed graph  $G(V, E)$  with  $n$  vertices. In order the algorithm to be applied, the weights of the edges  $f(i,j)$  must be *positive numbers*.
- The algorithm exists in many versions, the more common version defines a vertex as the "initial" vertex and finds the shortest distance from the initial vertex to all other vertices in the graph.

## Dijkstra's algorithm

- The algorithm can also be used to find the shortest distance from a vertex to the vertex of a destination, by stopping the algorithm after the shortest path to the vertex of the destination has been determined.
- *For example*, if the vertices of a graph represent cities and the endpoints represent the distance between two cities connected by a direct path, Dijkstra's algorithm can be used to find the shortest path between a city and all other cities.

# Dijkstra's algorithm

- Without applying any possible optimization, this algorithm has a complexity of  $O(N^2)$ , where  $N$  is the number of vertices in the given graph.
- If the algorithm is implemented with a possible optimization, it obtains a complexity of  $O(\log(N)*M)$ , where  $M$  is the number of edges in the graph, which is generally a better complexity, but it can also exceed  $N^2$ .
- Although it is the most efficient of the shortest path finding algorithms, this algorithm is not applicable to every graph. The graphs in which it is used must not have negative cycles (cycles in which the sum of the edge weights is less than 0).

# Dijkstra's algorithm

- **Example 1:** Consider a graph  $G = (V, E)$  whose vertices correspond to the airports in the world and whose arcs correspond to the air lines between them. How to choose a route to travel between two points so that this route (path) is optimal in some sense ?
- Optimality can be related to length, speed, security, etc.
- Let each arc of the graph be mapped to a weight  $c(x, y)$ , which is interpreted in terms of kilometers, time, etc. These weights can also be specified in a matrix  $C = (c_{ij})$ . The elements of the *weight matrix* can be positive, negative or zero.

## Dijkstra's algorithm

- **Example 2:** A commercial traveler plans to travel from Sofia to Varna, intending to visit other cities on the way. The trader knows with great precision what profit the possible visit of customers in the respective city will bring him. What route should this trader choose?
- Consider a graph  $G$  with vertices the cities that a trader is likely to visit.
- If the weight of each arc means "travel cost" - "expected revenue", it is clear that there are likely to be arcs with a negative weight (sections where transport costs are greater than expected profit).



# Dijkstra's algorithm

- In this case, the commercial traveler must choose a route corresponding to the shortest route in the graph between the vertices "Sofia" and "Varna. "
- We note that the *shortest path search algorithms* are different in the cases "*all weights are non-negative*" and the *weights are arbitrary*.
- Furthermore, in the absence of an arc  $(x, y)$  from the graph, we will consider its weight to be  $= \infty$ . We will call the weights of arcs and paths by the most natural term for the case - *lengths*.
- In the problem of finding the shortest paths, the constraint that there are no cycles with negative weight in the graph  $G$  is imposed. ERASMUS+



# Dijkstra's algorithm

## The idea of Dijkstra's algorithm

- In this algorithm, the lengths  $c_{ij}$  of all arcs are assumed to be non-negative.
- The algorithm can be considered as a process of sequentially marking the vertices of a graph with corresponding numbers.
- In the general case, the *marking number*  $d(x)$  at vertex  $x$  is temporary and gives an upper bound for the path length from  $s$  to  $x$ .
- During the algorithm's execution, the values of the marking numbers are decreased, with exactly one of the temporary marking numbers held *constant* (*colored*) at each step.

# Dijkstra's algorithm

## The idea of Dijkstra's algorithm

- In this case, the constant (colored) marking number  $d(x)$  is no longer some upper bound, but is the exact length of the shortest path from  $s$  to  $x$ .
- We also consider the corresponding vertex  $x$  to be colored.
- When except of shortest path length, the path itself is searched as well one of the arcs of the graph is also colored, thus including it in the searched path.

# Dijkstra's algorithm

## Steps of the Dijkstra algorithm

- **STEP 1.** Color the initial vertex  $s$ , put  
 $d(s) = 0$  (constant marking number),  
 $d(x) = \infty$  (temporary marking numbers),  $x \neq s$ ,  
 $p = s$  (the last colored vertex).
- **STEP 2.** (*Changing the temporary marking numbers.*) For all uncolored vertices  $x$ , recalculate  $d(x)$  using the formula:

$$d(x) = \min \{d(x), d(p) + c(p, x)\} \quad (1)$$

# Dijkstra's algorithm

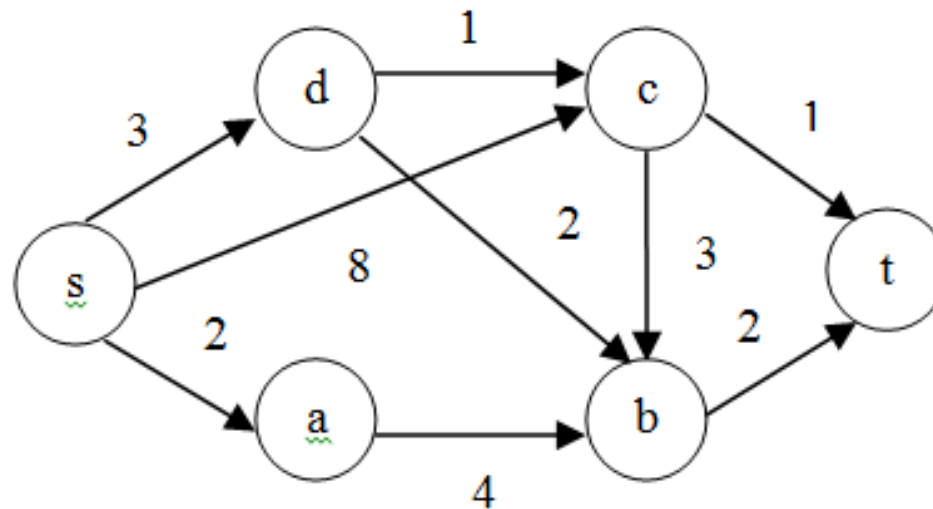
## Steps of the Dijkstra algorithm

If for any uncolored vertex  $x$ ,  $d(x) = \infty$ , terminate the procedure - there are no paths from  $s$  to the uncolored vertices in the graph. Otherwise, color the vertex  $x$  for which the weight  $d(x)$  is minimum. Also color the arc , entering in  $x$ , for which the minimum is reached. Put  $p = x$ .

- **STEP 3.** If  $p = t$ , end of procedure, the only path from  $s$  to  $t$  consisting of colored arcs is the shortest path between  $s$  and  $t$ . Otherwise, go to *step 2*.

# Dijkstra's algorithm

**Problem:** Find the shortest path between  $s$  and  $t$  in the following graph using Dijkstra's algorithm:



**STEP 1.** We assume  $d(s) = 0$  and  $d(x) = \infty$ , for all other vertices. We color  $s$ . We assume  $p = s$ .

## Dijkstra's algorithm

**STEP 2.** Using formula (1), we compute the new marking numbers for the uncolored vertices of the graph.

$$d(a) = \min \{d(a), d(s) + c(s, a)\} = \min \{\infty, 0 + 2\} = 2,$$

$$d(d) = \min \{d(d), d(s) + c(s, d)\} = \min \{\infty, 0 + 3\} = 3,$$

$$d(c) = \min \{d(c), d(s) + c(s, c)\} = \min \{\infty, 0 + 8\} = 8.$$

Since  $d(a)$  is the minimum marking number among the recomputed numbers, we *color the vertex  $a$  and the arc  $(s, a)$* . We assume  $p = a$ .

**STEP 3.** Forward us to step 2, since the vertex  $t$  is not colored.

# Dijkstra's algorithm

**STEP 2.**  $p = a$

$$d(b) = \min \{\infty, d(a) + c(a, b)\} = \min \{\infty, 2 + 4\} = 6,$$

$$d(d) = 3, d(c) = 8, d(t) = \infty.$$

The minimum number is  $d(d) = 3$ , so we *color the vertex  $d$  and the arc  $(s, d)$* . We assume  $p = d$ .

**STEP 3.** Forward us to step 2.

**STEP 2.**  $p = d$

$$d(c) = \min \{8, d(d) + c(d, c)\} = \min \{8, 3 + 1\} = 4,$$

$$d(b) = \min \{6, d(d) + c(d, b)\} = \min \{6, 3 + 2\} = 5,$$

$$d(t) = \infty.$$

*Color the vertex  $c$  and the arc  $(d, c)$ . We assume  $p = c$ .*

**STEP 3.** Forward us to step 2.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

## Dijkstra's algorithm

**STEP 2.**  $p = c$

$$d(b) = \min \{5, 4 + 3\} = 5,$$

$$d(t) = \min \{\infty, d(c) + c(c, t)\} = \min \{\infty, 4 + 1\} = 5.$$

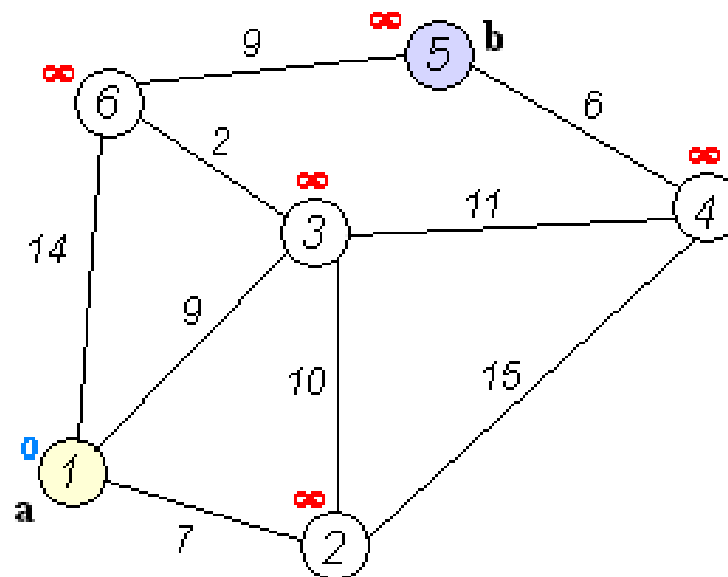
The minimum is any of the numbers  $d(b) = d(t) = 5$ . We choose  $d(t)$ , *color the vertex  $t$  and the arc  $(c, t)$* . We assume  $p = t$ .

**STEP 3.** End of algorithm. The tree  $(s, a), (s, d), (d, c), (c, t)$  of the shortest paths also gives the shortest  $(s - t)$  path  $(s, d), (d, c), (c, t)$  whose length is  $3 + 1 + 1 = 5$ .



# Dijkstra's algorithm

*Interactive example:*



# Bellman–Ford algorithm

- **The Bellman-Ford** algorithm is an algorithm that computes the shortest paths from one vertex to all other vertices in a directed, weighted graph. The Bellman-Ford algorithm is an example of **Dynamic Programming** and follows the bottom-up approach.
- It is slower than Dijkstra's algorithm , but much more flexible when traversing graphs whose edge weights are negative numbers.
- The algorithm is named after two of its creators, Richard Bellman and Lester Ford Jr, who published the algorithm in 1958 and 1956, respectively. Edward F. Moore also published the same algorithm in 1957 and for this reason it is sometimes found as the **Bellman-Ford-Moore algorithm**.

# Bellman–Ford algorithm

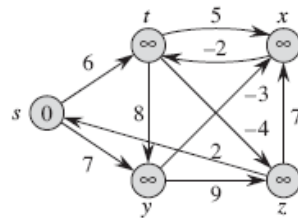
Bellman-Ford algorithm, is very rarely used in race programming, mainly when one has to search for a shortest path in a graph with negative edges. Here it is:

1. For each vertex adjacent to the initial vertex, the value of the shortest path for the moment is the edge next to it, and for the rest, infinity.
2. For each vertex in the graph, we check whether the current shortest path to it is not greater than the sum of the paths of each of the vertices from which it can be reached + the length of the edge between them.

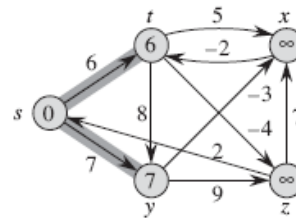
Step 2 is repeated  $N-2$  times, then we have the final shortest paths in the graph.

# Bellman–Ford algorithm

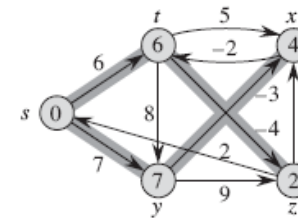
## Implementation of the Bellman-Ford algorithm for a 5-vertex graph



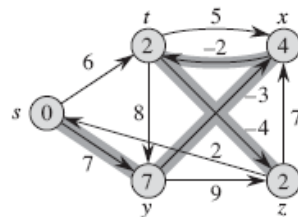
(a)



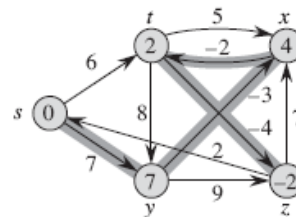
(b)



(c)



(d)



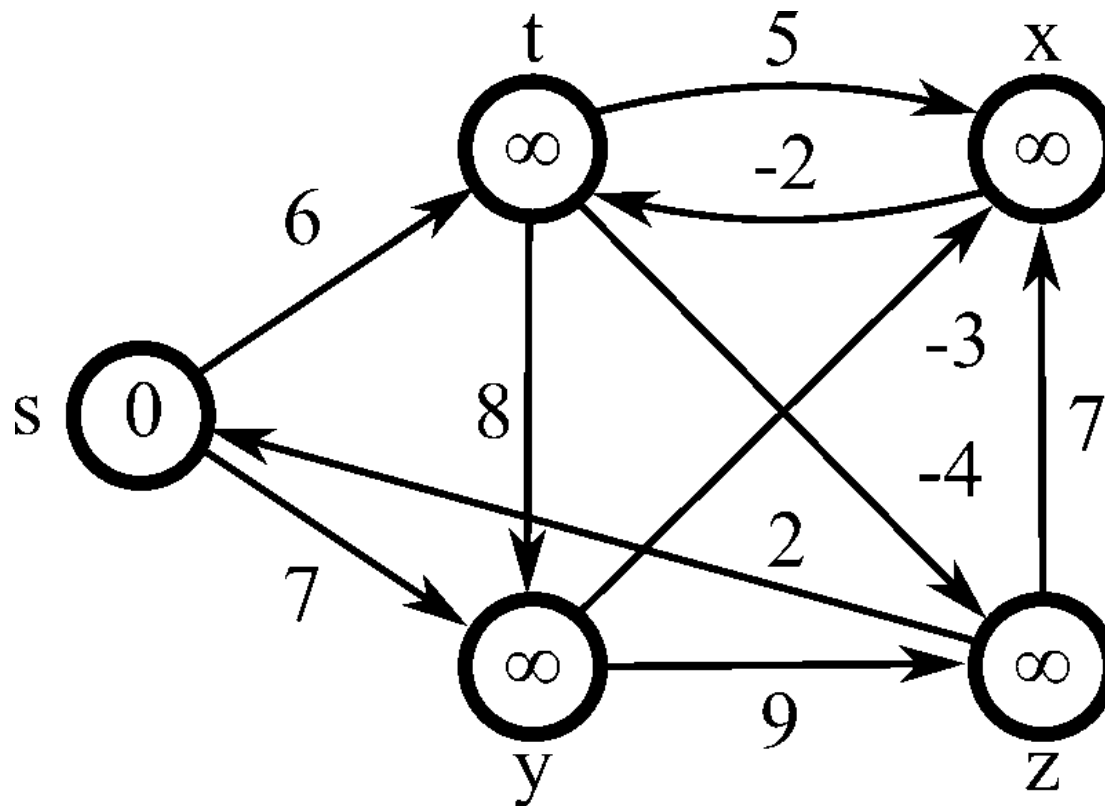
(e)

a) The situation right before the first crossing of the vertices.

(b) - (e) The situation after each subsequent passage through the vertices.

# Bellman–Ford algorithm

*Interactive example:*

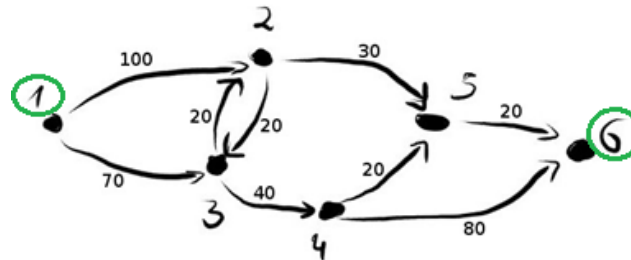


ERASMUS+

# Graph traversal

## application in DB

- Let's go back to the example from one of the previous lectures. Let's assume we've got a **database** with a list of nodes and a list of links between them (you can think of them as cities and roads). Our task is to find the shortest path from node 1 to node 6.



- we may have would be to get all rows from both tables and implement a **Dijkstra's** algorithm or **Bellman–Ford** algorithm with a single **SQL** query!

## Questions and exercises:

1. What is shortest path in a graph ?
2. Why does Dijkstra's Algorithm fail on negative weights ?
3. Why is Dijkstra's algorithm considered a greedy algorithm ?
4. What are the differences between Bellman Ford's and Dijkstra's algorithms?



# Thank you for your attention!



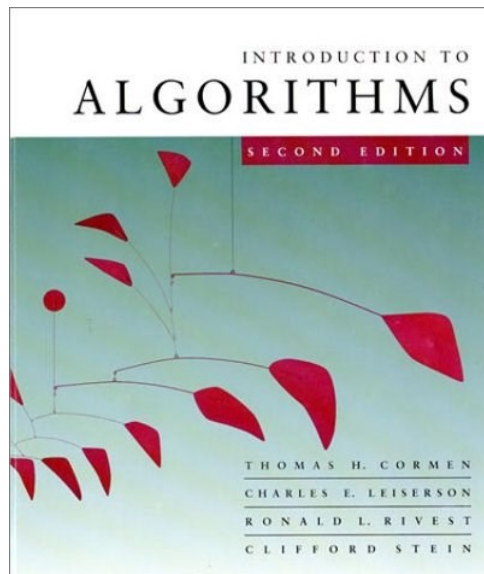
# ON-LINE DISTANCE COURSE ON DATABASES

- ❑ **Module 3. Algorithms and their applications in databases for query optimization**
- ❑ **Topic 4. Graph algorithms**
  - ❑ Lesson 4. Maximum flow in graph.



ERASMUS+

# MAXIMUM FLOW GRAPH



Carmen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2009) [1990]. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.



Preslav Nakov, Panayot Dobrikov (2002). Programming= ++Algorithms). Top Team Co, Sofia.

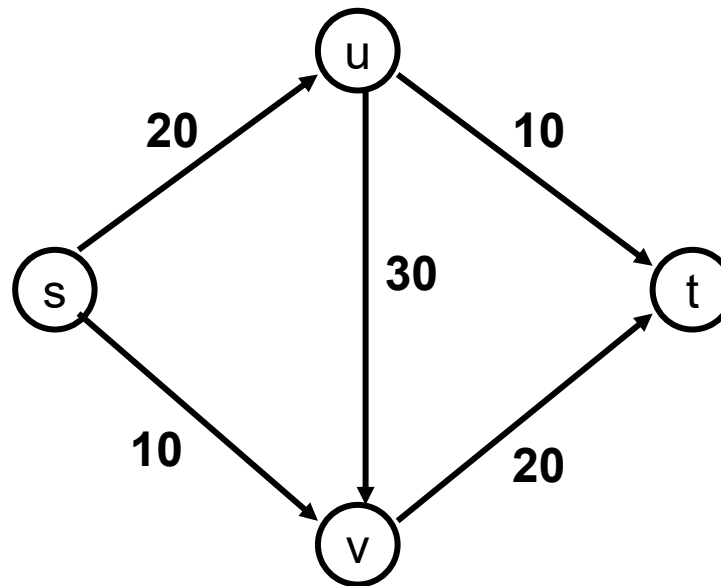
# Outline

- Network flow definitions
- Flow examples
- Augmenting Paths
- Residual Graph
- Ford Fulkerson Algorithm
- Cuts
- Maxflow-MinCut Theorem

# Network Flow Definitions

- Capacity
- Source, Sink
- Capacity Condition
- Conservation Condition
- Value of a flow

# Flow Example



## Introduction

- The **flow** sets a way to transfer objects from one vertex of a graph to another along its arcs (or edges).
- The initial vertex from which this transfer of quantities starts is called source and is usually denoted by  $s$ .
- The vertex to which this transfer has to be carried out is called sink (stock). The sink is usually denoted by  $t$ .
- *Consider an oriented graph.*

## Introduction

- We may consider it as a network of pipes in which a substance moves from a source to a sink.
- Object flows may be considered as the flow of electricity through wires, pipe pieces, information connections, or goods from the manufacturer to the consumer.
- Objects that move, "flow" from a source to a sink are called flow units or units only.
- The amount of flow units that can pass through the arc is called a *capacity (throughput)*.
- We can use the max flow algorithm in **SQL** query optimization to reduce network traffic costs or avoid timeout errors.

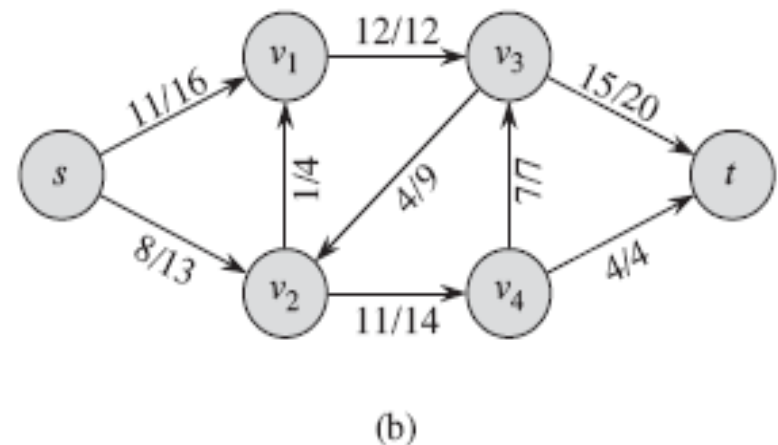
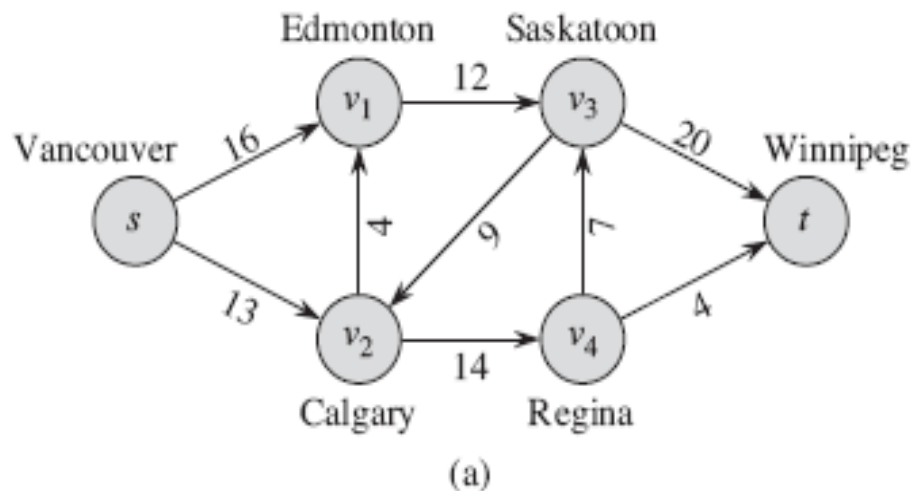
## Flow networks

- A network is a graph  $G=(V, E)$  in which each arc is assigned some throughput (capacity).
- Let's consider a flow network in an oriented graph  $G=(V,E)$ , each edge  $(u,v) \in E$  of which is denoted by a number  $c(u,v) \geq 0$  called throughput (capacity).
- In the case  $(u, v) \notin E$ . We assume  $c(u,v)=0$ . We set two vertices in the graph: source  $s$  and sink  $t$ .



# Flow networks

- For convenience, we can assume that each vertex  $v \in V$  lies somehow on the path  $s \rightarrow v \rightarrow t$  from source to sink. In this case, the graph also connects  $|E| \geq |V| - 1$ .
- The figure below shows an example:



# Flow networks

- In the figure above we see:
  - (a) *The flow in  $G=(V,E)$ , describes the possibility of transporting goods from the source factory  $s$  in Vancouver to a sink  $t$  - warehouse in Winnipeg. On each edge is written the maximum number of boxes that can be sent per day.*
  - (b)  *$f$  flow in the network  $G$  with value  $|f|= 19$ . Flow/capacity are indicated on each arc.*
- *The value  $|f|$  of the flow  $f$  is defined as:*

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) ,$$

## Ford-Fulkerson method

- The Ford–Fulkerson **method** or Ford–Fulkerson algorithm (FFA) is a **greedy algorithm** that computes the maximum flow in a flow network.
- We are talking about method, not the algorithm, as there are several algorithms for implementing this method; they differ during operation.
- Three notions play a key role in the Ford-Fulkerson method: residual network, complementary path and cut.
- Finding the maximum flow by Ford-Fulkerson method is performed step by step.
- At the beginning the flow is zero (and its value equals zero).

## Ford-Fulkerson method

- At each step we increase the value of the flow. In order to do this, we find an augmenting path in which we can skip many more "substances" , and use it to increase the flow.
- This step is repeated until there are augmenting path.
- This shows the Max-flow min-cut theorem.

# Ford-Fulkerson method

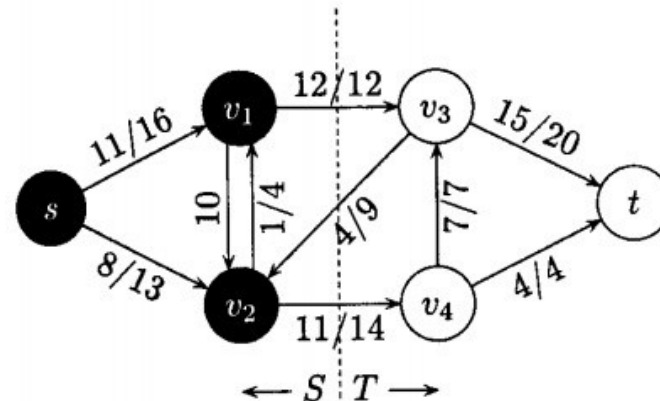
- **Residual networks**
- Let a network and a flow in it be given. Informally, the residual network represents the stream network, but with only the edges remaining along which flow can still be passed.
- The residual network is defined on the same graph  $G(V, E)$ , but a new function is introduced - **residual capacity**, :  $cf(u, v) = c(u, v) - f(u, v)$ . Then  $Gf(V, E')$ , where  $E'$  are those edges for which  $cf(u, v) > 0$ . We will call them **residual capacity edges**.

## Ford-Fulkerson method

- **Network cuts**
- The Ford - Fulkerson method adds successive flows along the augmenting path until the maximum flow is obtained.
- According to the maximum flow and minimum cut theorem, the flow is maximum if and when the residual network does not contain an augmenting path.
- The minimum cut is the one with the lowest throughput (among all cuts in the network).

# Ford-Fulkerson method

- **Network cuts**
- The figure below shows a network cut.  
( $\{s, v_1, v_2\}, \{v_3, v_4, t\}$ )



- The flow through this cut is:

$$f(v_1, v_3) + f(v_2, v_3) + f(v_2, v_4) = 12 + (-4) + 11 = 9$$

and the throughput of the cut is:

$$c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26$$

# Ford-Fulkerson method

- **Network cuts**
- Cut  $(S, T)$  in the network from the figure above.
- Here  $S = \{s, v_1, v_2\}$  (the black vertices) and  $T = \{v_3, v_4, t\}$  (the white vertices).
- At  $f(S, T) = 19$  (flow through the cut) and  $s(S, T) = 26$  (throughput)
- As can be seen, the flow through the cut as opposed to the throughput of the cut can include negative terms as well.



# Ford-Fulkerson method

- **General scheme of the Ford-Fulkerson algorithm**
- The Ford-Fulkerson method works by choosing an arbitrary increasing path  $p$  and increase the flow  $f$  at each step.
- Add the flow value  $C_f(p)$  to the path  $p$ .
- This algorithm uses an array  $f[u, v]$  to store the current flow values.
- $c(u, v) = 0$ , if  $(u, v)$  doesn't belong to  $E$ .
- The running time of the Ford-Fulkerson procedure depends on how many times it should be run.

# Ford-Fulkerson method

- In principle, the algorithm may not stop at all if the flow value continues to increase in smaller steps without reaching a maximum.
- However, it is also possible to run the algorithm in polynomial time.
- The Ford-Fulkerson algorithm is good because it is faster and significantly less complex than other network flow algorithms.
- It belongs to the group of optimization methods and was created in 1956.

## Ford-Fulkerson method

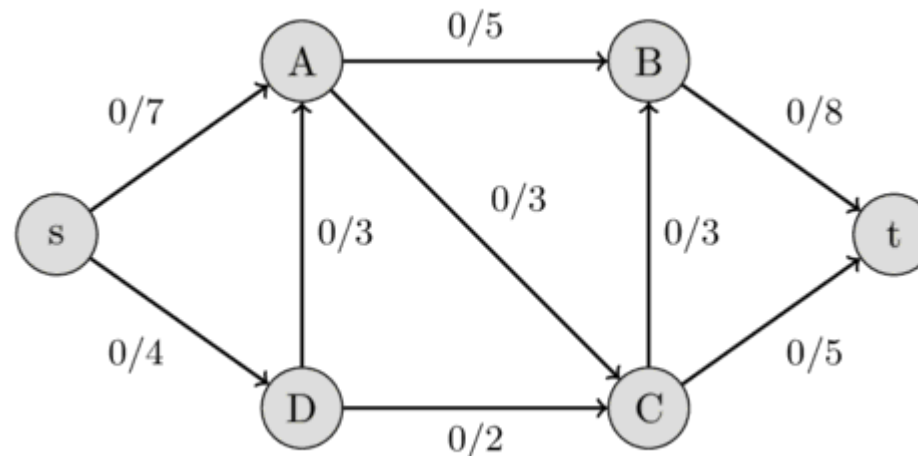
- Other possible solutions are the **Edmonds-Karp** algorithm, which removes the dependence of the Ford-Fulkerson algorithm on the magnitude of the capacities, the Goldberg algorithms, Onaga, etc.
- Practically, for significantly large networks, there is no algorithm that can derive the maximum throughput in an acceptable amount of time.
- The max-flow min-cut theorem gives us a reason to use the Ford-Fulkerson algorithm to find the numerical value of the minimum cut.

## Ford-Fulkerson method

- *But an important question is also where, in fact, is it located (minimum cut)?*
- If it turns out that the traffic needed between two nodes of the network is greater than the capacity of the lines connecting them, the finding will show which are the lines that limit it (it may be that due to unused reserves only one line needs to be expanded) and will therefore answer the question which lines should be optimized.

# Ford-Fulkerson method

- *EXAMPLE:*
- The following image shows a flow network. The first value of each edge represents the flow, which is initially 0, and the second value represents the capacity.

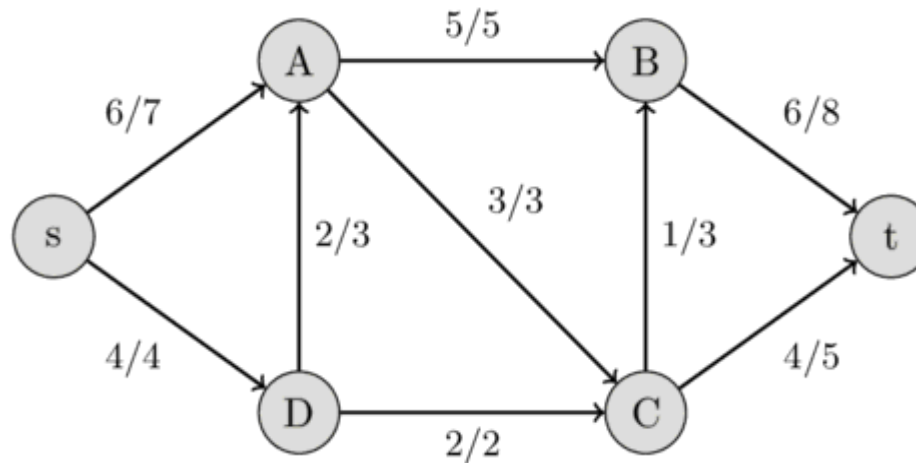


## Ford-Fulkerson method

- *EXAMPLE:*
- The value of the flow of a network is the sum of all the flows that get produced in the source  $s$ , or equivalently to the sum of all the flows that are consumed by the sink  $t$ . A maximal flow is a flow with the maximal possible value. Finding this maximal flow of a flow network is the problem that we want to solve.
- In the visualization with water pipes, the problem can be formulated in the following way: how much water can we push through the pipes from the source to the sink?

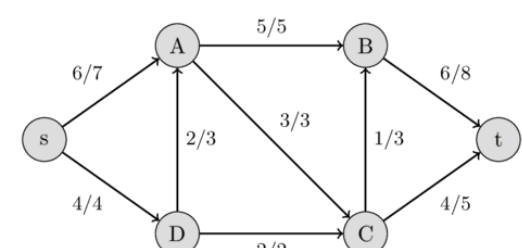
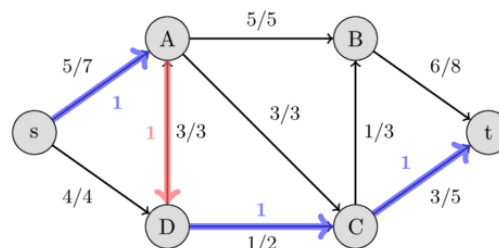
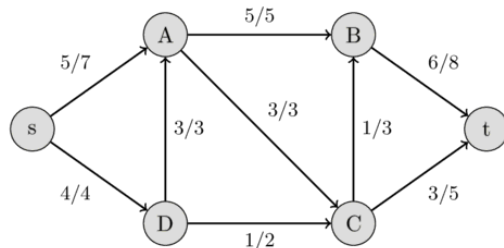
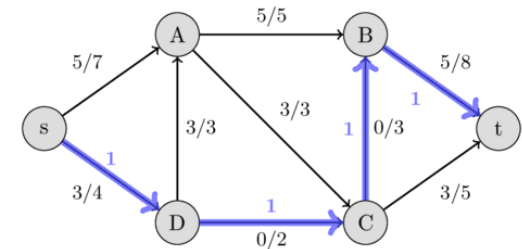
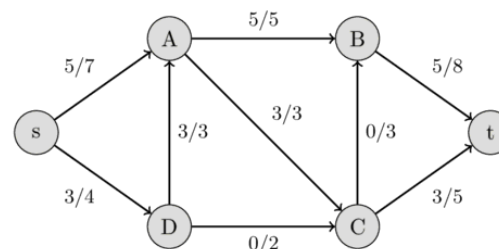
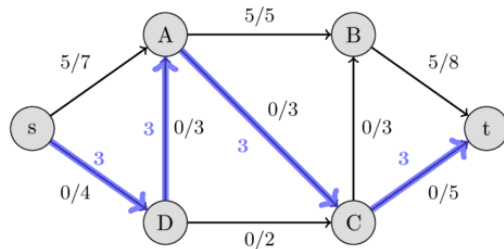
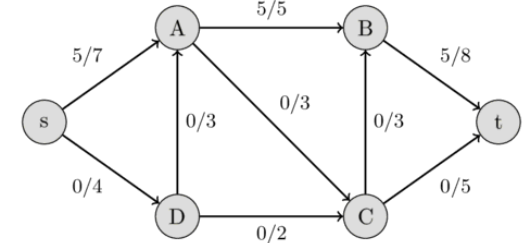
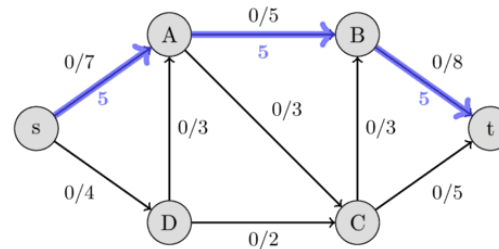
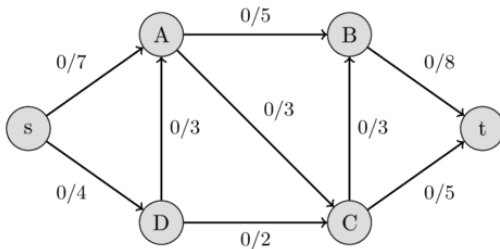
# Ford-Fulkerson method

- *EXAMPLE:*
- The following image shows the maximal flow in the flow network:



# Ford-Fulkerson method

## • EXAMPLE:



10 is the maximal flow

ERASMUS+



# Ford-Fulkerson method

## more than one source and consumer

- If more than one source and consumer are given in the problem, we can easily reduce it to the one just discussed above.
- Let the sources be  $s_1, s_2, \dots, s_p$ , and consumers  $t_1, t_2, \dots, t_q$ .
- We add a new vertex  $s$  in the graph called supersource and a new vertex  $t$ , the superconsumer.
- In addition to these two new vertices, we also add the edges:
  - $(s, s_i)$ , such that  $c(s, s_i) = +\infty$ , for  $i = 1, 2, \dots, p$ .
  - $(t_j, t)$ , such that  $c(t_j, t) = +\infty$ , for  $j = 1, 2, \dots, q$ .

# Ford-Fulkerson method

## more than one source and consumer

- Using the Ford-Fulkerson algorithm, we can find a maximum flow from  $s$  to  $t$  that will be equal to the one sought in the original version of the problem.
- It is possible to complicate the problem by placing a restriction on the flow through each vertex.

Let be given a function  $v(i)$ ,  $v(i) \geq 0$ ,  $i \in V$  such that

$$\sum_{j \in V} f(i, j) \leq v(i), i \in V$$

# Ford-Fulkerson method

## more than one source and consumer

- We are looking for the maximum flow from the source  $s$  to the consumer  $t$ . Again, we can solve the problem by reducing it to the standard maximum flow search problem. For this purpose we will construct a new graph  $G'(V', E')$  where:
  - 1) on each vertex  $i \in V$  correspond two vertices  $v': i_1$  and  $i_2$  that will be connected by an edge  $(i_1, i_2)$ . The throughput  $c(i_1, i_2)$  of this edge will be equal  $v(i)$ .
  - 2) each edge  $(i, j)$  from  $G$  is transferred to  $G'$  as the edge  $(i_2, j_1)$ .
  - 3) In the new graph  $G'$  there won't be a function  $v$  restricting the flow through the vertices.
- As in the previous paragraph, the maximum flow in  $G'$  found by the Ford-Fulkerson algorithm will be maximum for the graph  $G$  as well.

## Questions and exercises:

1. What does Maximum flow problem involve ?
2. A network can have only one source and one sink  
**! True or false ?**
3. What is the source ?
4. Which algorithm is used to solve a maximum flow problem?
5. Ford-Fulkerson algorithm is used to solve a maximum flow problem ! True or false ?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 4. Biological Databases

### ☐ Topic 1. Amino acids, peptides and proteins

#### ☐ Lesson 1. Amino acids

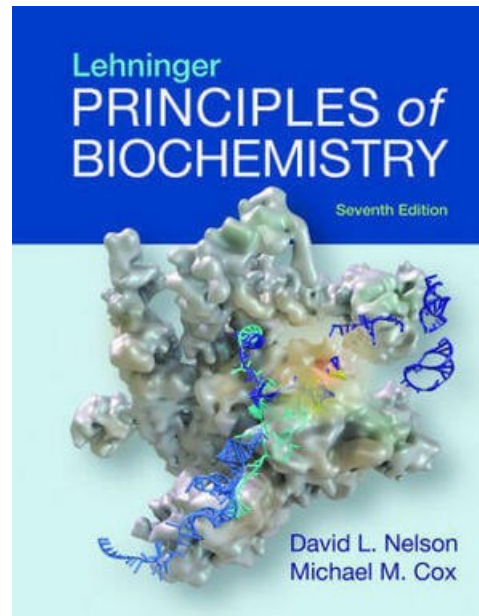


ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# AMINO ACIDS



Lehninger, A. L., Nelson, D. L., & Cox, M. M.  
(2000). *Lehninger principles of biochemistry*.  
New York: Worth Publishers.



# Contents

- Introduction
- Physical Properties
- Chemical Properties
- Structure of Amino acids
- Classification of amino acids
- Functions of Amino acids

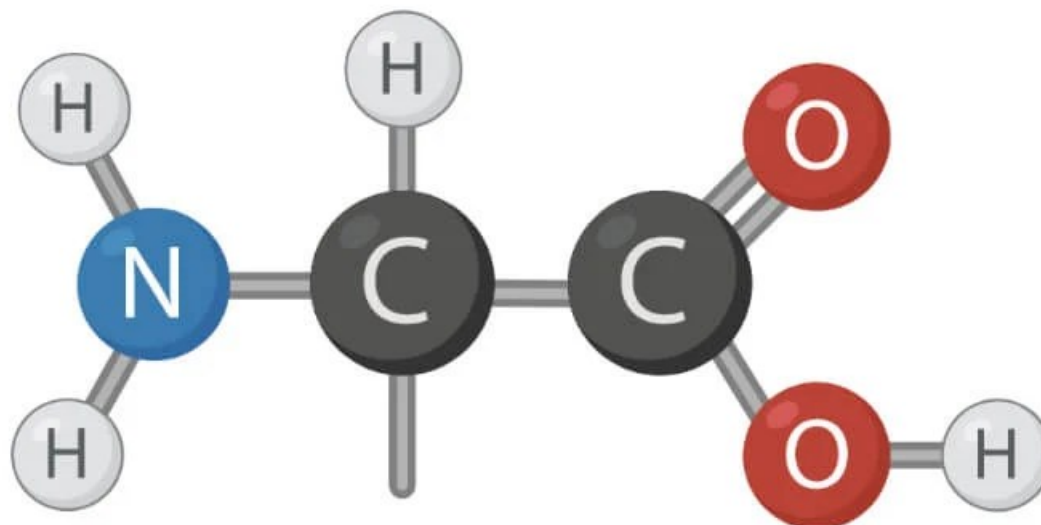




# Introduction

Amino acids constitute a group of neutral products clearly distinguished from other natural compounds chemically, mainly because of their ampholytic properties, and biochemically, mainly because of their role as protein constituents. An amino acid is a carboxylic acid-containing an aliphatic primary amino group in the  $\alpha$  position to the carboxyl group and with a characteristic stereochemistry. Proteins are biosynthesized from 20 amino acids in a system involving strict genetic control. Thus, amino acids are the basic unit of proteins. More than 300 amino acids are found in nature but only 20 amino acids are standard and present in protein because they are coded by genes. Other amino acids are modified amino acids and are called non-protein amino acids. Some are residues modified after a protein has been synthesized by posttranslational modifications; others are amino acids present in living organisms but not as constituents of proteins.

## Amino Acids- Properties, Structure, Classification, Functions



# Physical Properties

- Amino acids are colorless, crystalline solid.
- All amino acids have a high melting point greater than 200o
- Solubility: They are soluble in water, slightly soluble in alcohol, and dissolve with difficulty in methanol, ethanol, and propanol. R-group of amino acids and pH of the solvent play important role in solubility.
- On heating to high temperatures, they decompose.
- All amino acids (except glycine) are optically active.
- Peptide bond formation: Amino acids can connect with a peptide bond involving their amino and carboxylate groups. A covalent bond formed between the alpha-amino group of one amino acid and an alpha-carboxyl group of other forming -CO-NH-linkage. Peptide bonds are planar and partially ionic.

# Chemical Properties

- Zwitterionic property

A zwitterion is a molecule with functional groups, of which at least one has a positive and one has a negative electrical charge. The net charge of the entire molecule is zero. Amino acids are the best-known examples of zwitterions. They contain an amine group (basic) and a carboxylic group (acidic). The  $\text{-NH}_2$  group is the stronger base, and so it picks up  $\text{H}^+$  from the  $\text{-COOH}$  group to leave a zwitterion. The (neutral) zwitterion is the usual form of amino acids that exist in the solution.

- Amphoteric property

Amino acids are amphoteric in nature that is they act as both acids and base due to the two amine and carboxylic groups present.

# Chemical Properties

- Ninhydrin test

When 1 ml of Ninhydrin solution is added to a 1 ml protein solution and heated, the formation of a violet color indicates the presence of  $\alpha$ -amino acids.

- Xanthoproteic test

The xanthoproteic test is performed for the detection of aromatic amino acids (tyrosine, tryptophan, and phenylalanine) in a protein solution. The nitration of benzoid radicals present in the amino acid chain occurs due to a reaction with nitric acid, giving the solution yellow coloration.

# Chemical Properties

- Reaction with Sanger's reagent

Sanger's reagent (1-fluoro-2, 4-dinitrobenzene) reacts with a free amino group in the peptide chain in a mild alkaline medium under cold conditions.

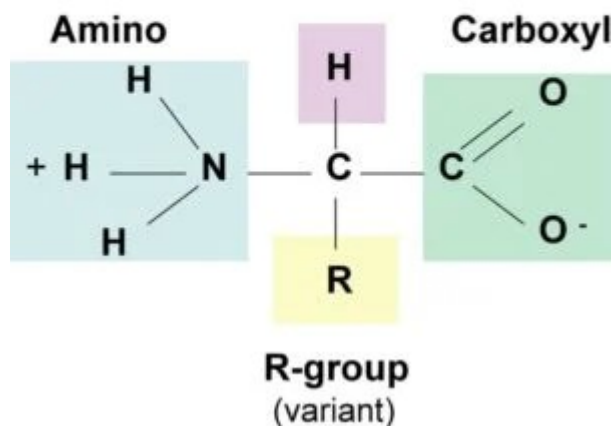
- Reaction with nitrous acid

Nitrous acid reacts with the amino group to liberate nitrogen and form the corresponding hydroxyl.

# Structure of Amino acids

## Amino Acid Structure

Hydrogen



All 20 of the common amino acids are alpha-amino acids. They contain a carboxyl group, an amino group, and a side chain (R group), all attached to the  $\alpha$ -carbon.

Exceptions are:

- Glycine, which does not have a side chain. Its  $\alpha$ -carbon contains two hydrogens.
- Proline, in which the nitrogen is part of a ring.
- Thus, each amino acid has an amine group at one end and an acid group at the other, and a distinctive side chain. The backbone is the same for all amino acids while the side chain differs from one amino acid to the next.
- All of the 20 amino acids except glycine are of the L-configuration, as for all but one amino acid the  $\alpha$ -carbon is an asymmetric carbon. Because glycine does not contain an asymmetric carbon atom, it is not optically active and, thus, is neither D nor L.



# Classification of amino acids on the basis of R-group

## Amino Acid Chart

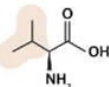
### Non-polar side chains



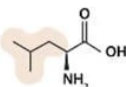
Glycine



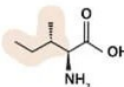
Alanine



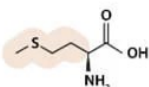
Valine



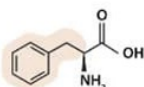
Leucine



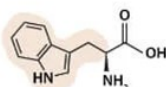
Isoleucine



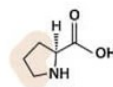
Methionine



Phenylalanine

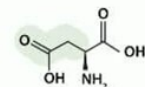


Tryptophan

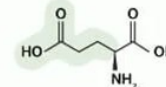


Proline

### Electrically charged side chains

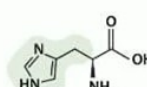


Aspartate

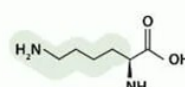


Glutamate

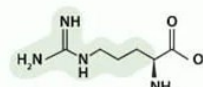
#### Acidic



Histidine



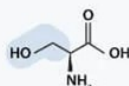
Lysine



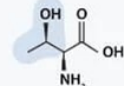
Arginine

#### Basic

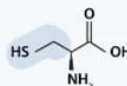
### Polar side chains



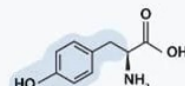
Serine



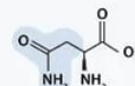
Threonine



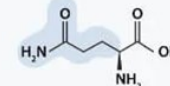
Cysteine



Tyrosine



Asparagine



Glutamine



# Classification of amino acids on the basis of R-group

1. **Nonpolar, Aliphatic amino acids:** The R groups in this class of amino acids are nonpolar and hydrophobic. Glycine, Alanine, Valine, leucine, Isoleucine, Methionine, Proline.
2. **Aromatic amino acids:** Phenylalanine, tyrosine, and tryptophan, with their aromatic side chains, are relatively nonpolar (hydrophobic). All can participate in hydrophobic interactions.
3. **Polar, Uncharged amino acids:** The R groups of these amino acids are more soluble in water, or more hydrophilic, than those of the nonpolar amino acids, because they contain functional groups that form hydrogen bonds with water. This class of amino acids includes serine, threonine, cysteine, asparagine, and glutamine.
4. **Acidic amino acids:** Amino acids in which R-group is acidic or negatively charged. Glutamic acid and Aspartic acid
5. **Basic amino acids:** Amino acids in which R-group is basic or positively charged. Lysine, Arginine, Histidine

# Classification of amino acids on the basis of nutrition

Essential	Conditionally Non-Essential	Non-Essential
Histidine	Arginine	Alanine
Isoleucine	Cystine	Asparagine
Leucine	Glutamine	Aspartate
Lysine	Glycine	Glutamate
Methionine	Proline	Serine
Phenylalanine	Tyrosine	
Threonine		
Tryptophan		
Valine		

# Classification of amino acids on the basis of nutrition

## Essential amino acids (Nine)

Nine amino acids cannot be synthesized in the body and, therefore, must be present in the diet in order for protein synthesis to occur.

These essential amino acids are histidine, isoleucine, leucine, lysine, methionine, phenylalanine, threonine, tryptophan, and valine.

## Non-essential amino acids (Eleven)

These amino acids can be synthesized in the body itself and hence do not necessarily need to be acquired through diet.

These non-essential amino acids are Arginine, glutamine, tyrosine, cysteine, glycine, proline, serine, ornithine, alanine, asparagine, and aspartate.

# Classification of amino acids on the basis of the metabolic fate

Glucogenic amino acids	Glucogenic and ketogenic	Ketogenic amino acids
Alanine, Arginine, Asparagine, Aspartate Asparagine, Cysteine, Methionine Glutamate, Glutamine, Glycine, Histidine Proline, Serine, Threonine, Valine	Tyrosine Isoleucine Phenylalanine Tryptophan	Leucine Lysine

# Classification of amino acids on the basis of the metabolic fate

1. **Glucogenic amino acids:** These amino acids serve as precursors of gluconeogenesis for glucose formation. Glycine, alanine, serine, aspartic acid, asparagine, glutamic acid, glutamine, proline, valine, methionine, cysteine, histidine, and arginine.
2. **Ketogenic amino acids:** These amino acids break down to form ketone bodies. Leucine and Lysine.
3. **Both glucogenic and ketogenic amino acids:** These amino acids break down to form precursors for both ketone bodies and glucose. Isoleucine, Phenylalanine, Tryptophan, and tyrosine.

# Functions of Amino acids

1. In particular, 20 very important amino acids are crucial for life as they contain peptides and proteins and are known to be the building blocks for all living things.
2. The linear sequence of amino acid residues in a polypeptide chain determines the three-dimensional configuration of a protein, and the structure of a protein determines its function.
3. Amino acids are imperative for sustaining the health of the human body. They largely promote the:

## Production of hormones

- Structure of muscles
- Human nervous system's healthy functioning
- The health of vital organs
- Normal cellular structure

# Functions of Amino acids

4. The amino acids are used by various tissues to synthesize proteins and to produce nitrogen-containing compounds (e.g., purines, heme, creatine, epinephrine), or they are oxidized to produce energy.
5. The breakdown of both dietary and tissue proteins yields nitrogen-containing substrates and carbon skeletons.
6. The nitrogen-containing substrates are used in the biosynthesis of purines, pyrimidines, neurotransmitters, hormones, porphyrins, and nonessential amino acids.
7. The carbon skeletons are used as a fuel source in the citric acid cycle, used for gluconeogenesis, or used in fatty acid synthesis.



# Review Questions

- Explain the physical and chemical properties of amino acids.
- What are the most popular classifications of amino acids?





Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 4. Biological Databases

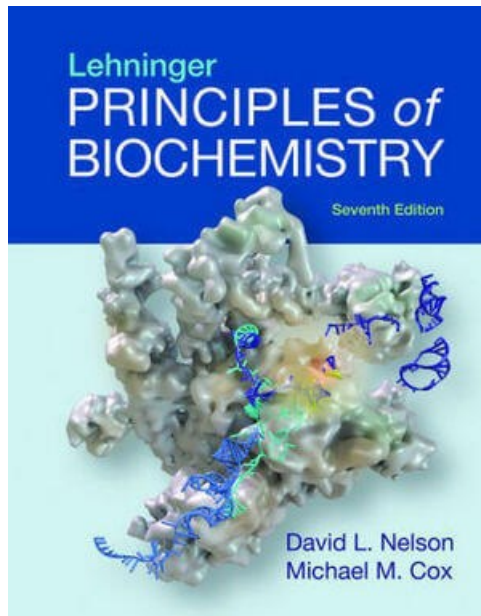
### ☐ Topic 1. Amino acids, peptides and proteins

#### ☐ Lesson 2. Peptides and proteins



ERASMUS+

# PEPTIDES AND PROTEINS



Lehninger, A. L., Nelson, D. L., & Cox, M. M.  
(2000). Lehninger principles of biochemistry.  
New York: Worth Publishers.



Molecular Biology of the Cell. 4th edition.  
Alberts B, Johnson A, Lewis J, et al.  
New York: Garland Science; 2002.



# Contents

- Introduction
- Primary Structure
- Secondary Structure
- Tertiary Structure
- Quaternary Structure



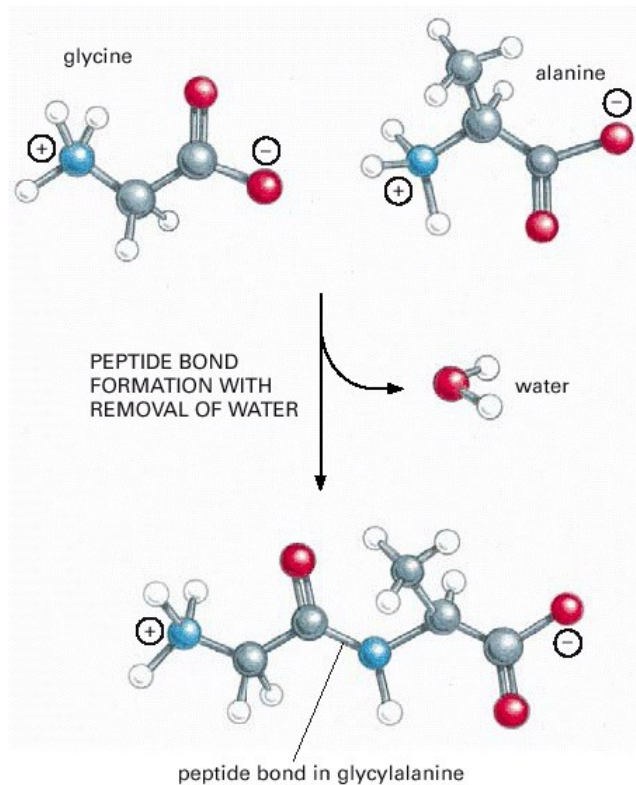
# Introduction

From a chemical point of view, proteins are by far the most structurally complex and functionally sophisticated molecules known. This is perhaps not surprising, once one realizes that the structure and chemistry of each protein has been developed and fine-tuned over billions of years of evolutionary history. We start this chapter by considering how the location of each amino acid in the long string of amino acids that forms a protein determines its three-dimensional shape. We will then use this understanding of protein structure at the atomic level to describe how the precise shape of each protein molecule determines its function in a cell.

# The Shape of a Protein Is Specified by Its Amino Acid Sequence

## A peptide bond

This covalent bond forms when the carbon atom from the carboxyl group of one amino acid shares electrons with the nitrogen atom (blue) from the amino group of a second amino acid. As indicated, a molecule of water is lost in this condensation reaction.

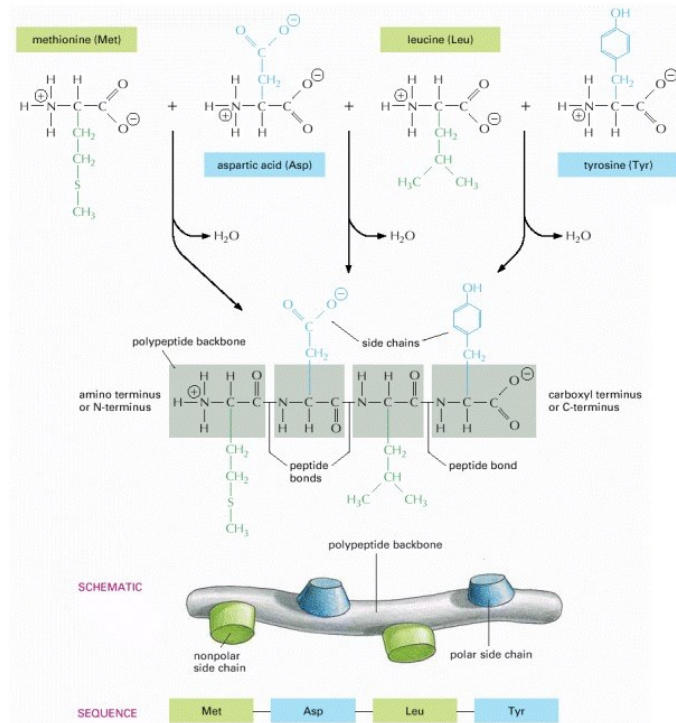




# The Shape of a Protein Is Specified by Its Amino Acid Sequence

The structural components of a protein

A protein consists of a polypeptide backbone with attached side chains. Each type of protein differs in its sequence and number of amino acids; therefore, it is the sequence of the chemically different side chains that makes each protein distinct. The two ends of a polypeptide chain are chemically different: the end carrying the free amino group ( $\text{NH}_3^+$ , also written  $\text{NH}_2$ ) is the amino terminus, or N-terminus, and that carrying the free carboxyl group ( $\text{COO}^-$ , also written  $\text{COOH}$ ) is the carboxyl terminus or C-terminus. The amino acid sequence of a protein is always presented in the N-to-C direction, reading from left to right..



# The Shape of a Protein Is Specified by Its Amino Acid Sequence

AMINO ACID			SIDE CHAIN
Aspartic acid	Asp	D	negative
Glutamic acid	Glu	E	negative
Arginine	Arg	R	positive
Lysine	Lys	K	positive
Histidine	His	H	positive
Asparagine	Asn	N	uncharged polar
Glutamine	Gln	Q	uncharged polar
Serine	Ser	S	uncharged polar
Threonine	Thr	T	uncharged polar
Tyrosine	Tyr	Y	uncharged polar

POLAR AMINO ACIDS

AMINO ACID			SIDE CHAIN
Alanine	Ala	A	nonpolar
Glycine	Gly	G	nonpolar
Valine	Val	V	nonpolar
Leucine	Leu	L	nonpolar
Isoleucine	Ile	I	nonpolar
Proline	Pro	P	nonpolar
Phenylalanine	Phe	F	nonpolar
Methionine	Met	M	nonpolar
Tryptophan	Trp	W	nonpolar
Cysteine	Cys	C	nonpolar

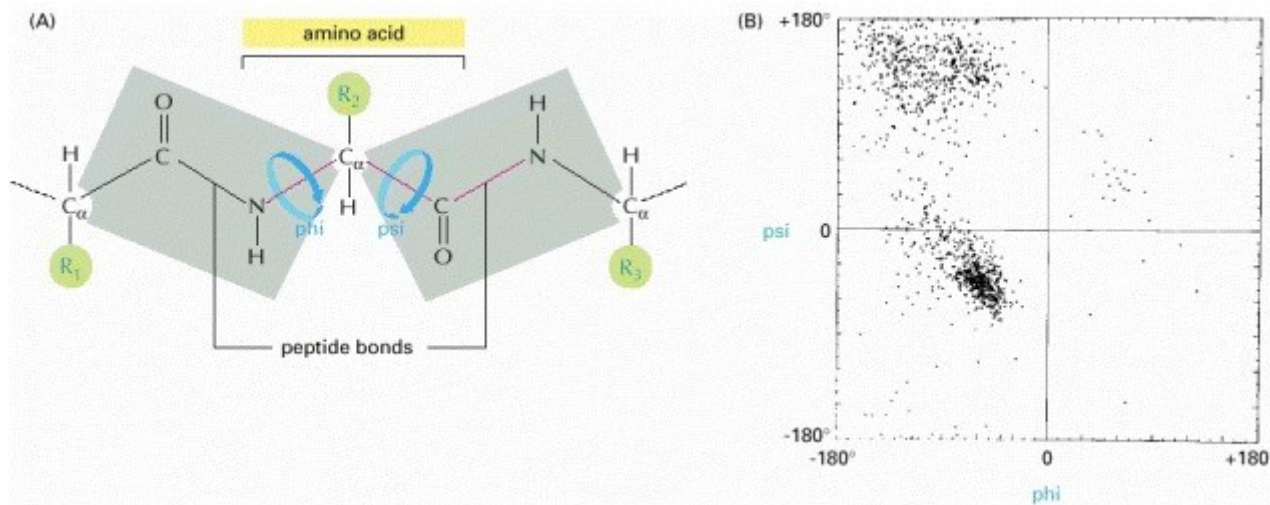
NONPOLAR AMINO ACIDS

The 20 amino acids found in proteins

Both three-letter and one-letter abbreviations are listed. As shown, there are equal numbers of polar and nonpolar side chains. For their atomic structures,



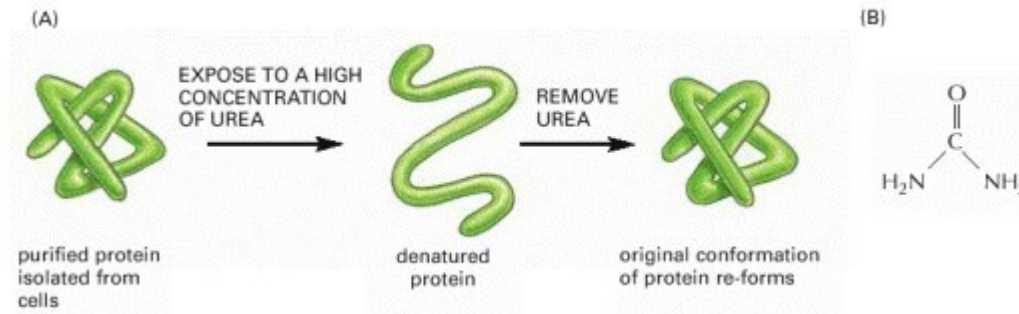
# The Shape of a Protein Is Specified by Its Amino Acid Sequence



## Steric limitations on the bond angles in a polypeptide chain

(A) Each amino acid contributes three bonds (red) to the backbone of the chain. The peptide bond is planar (gray shading) and does not permit rotation. By contrast, rotation can occur about the  $C_\alpha-C$  bond, whose angle of rotation is called psi ( $\psi$ ), and about the  $N-C_\alpha$  bond, whose angle of rotation is called phi ( $\phi$ ). By convention, an R group is often used to denote an amino acid side chain (green circles). (B) The conformation of the main-chain atoms in a protein is determined by one pair of  $\phi$  and  $\psi$  angles for each amino acid; because of steric collisions between atoms within each amino acid, most pairs of  $\phi$  and  $\psi$  angles do not occur. In this so-called Ramachandran plot, each dot represents an observed pair of angles in a protein.

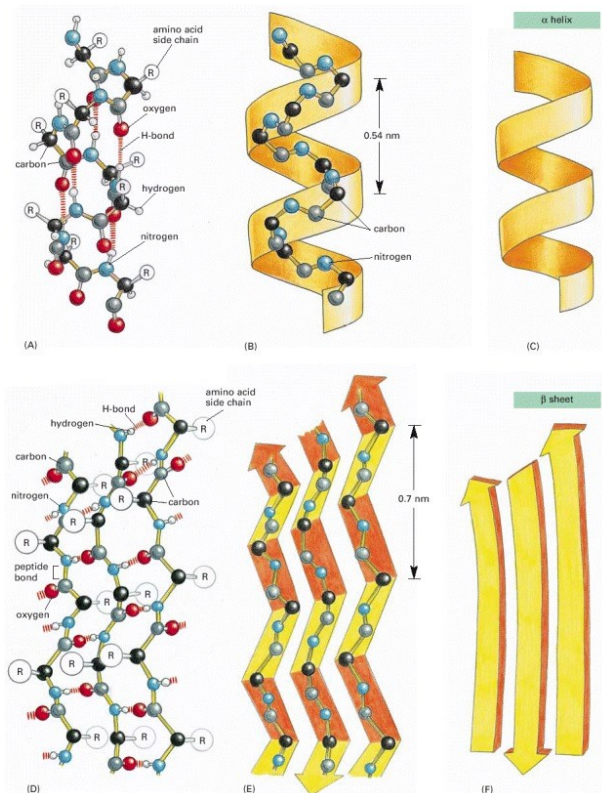
# Proteins Fold into a Conformation of Lowest Energy



## The refolding of a denatured protein

(A) This experiment demonstrates that the conformation of a protein is determined solely by its amino acid sequence. (B) The structure of urea. Urea is very soluble in water and unfolds proteins at high concentrations, where there is about one urea molecule for every six water molecules.

# The $\alpha$ Helix and the $\beta$ Sheet Are Common Folding Patterns



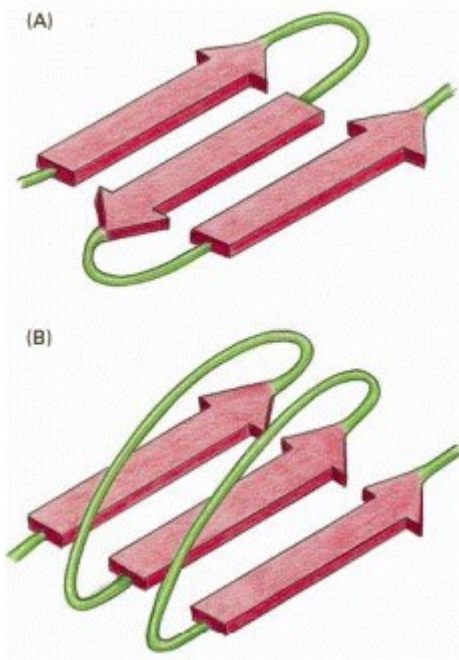
The regular conformation of the polypeptide backbone observed in the  $\alpha$  helix and the  $\beta$  sheet

(A, B, and C) The  $\alpha$  helix. The N–H of every peptide bond is hydrogen-bonded to the C=O of a neighboring peptide bond located four peptide bonds away in the same chain. (D, E, and F) The  $\beta$  sheet. In this example, adjacent peptide chains run in opposite (antiparallel) directions. The individual polypeptide chains (strands) in a  $\beta$  sheet are held together by hydrogen-bonding between peptide bonds in different strands, and the amino acid side chains in each strand alternately project above and below the plane of the sheet. (A) and (D) show all the atoms in the polypeptide backbone, but the amino acid side chains are truncated and denoted by R. In contrast, (B) and (E) show the backbone atoms only, while (C) and (F) display the shorthand symbols that are used to represent the  $\alpha$  helix and the  $\beta$  sheet in ribbon drawings of proteins

# The $\alpha$ Helix and the $\beta$ Sheet Are Common Folding Patterns

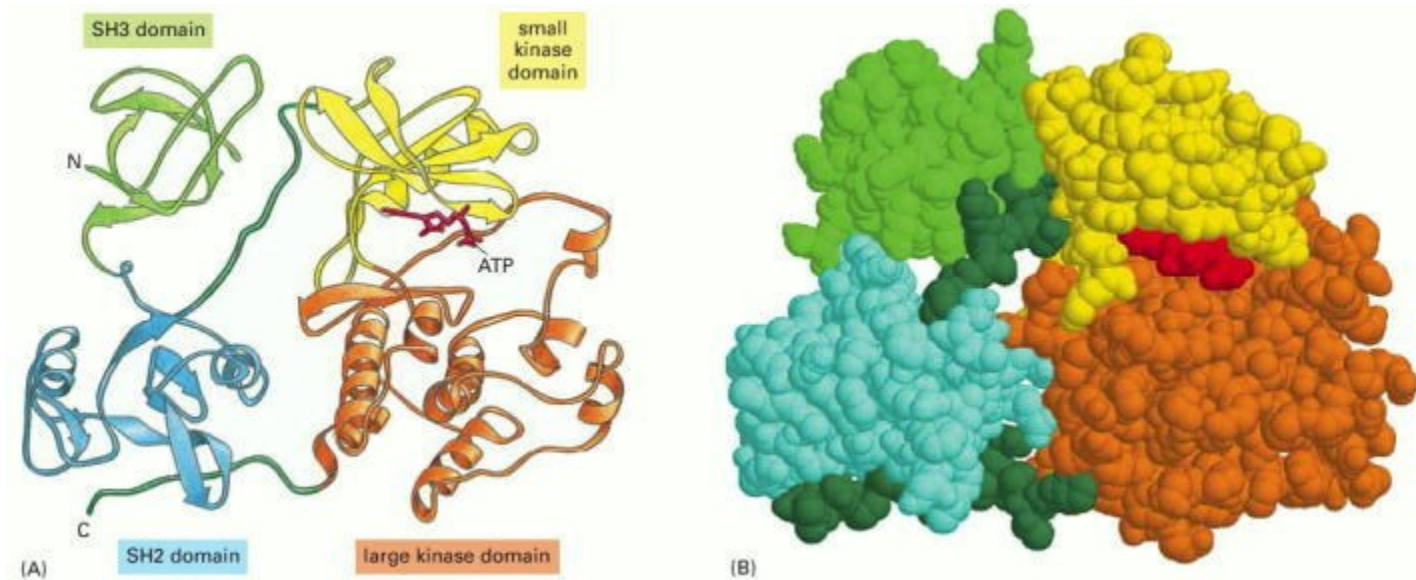
Two types of  $\beta$  sheet structures

(A) An antiparallel  $\beta$  sheet (see Figure 3-9D). (B) A parallel  $\beta$  sheet. Both of these structures are common in proteins.





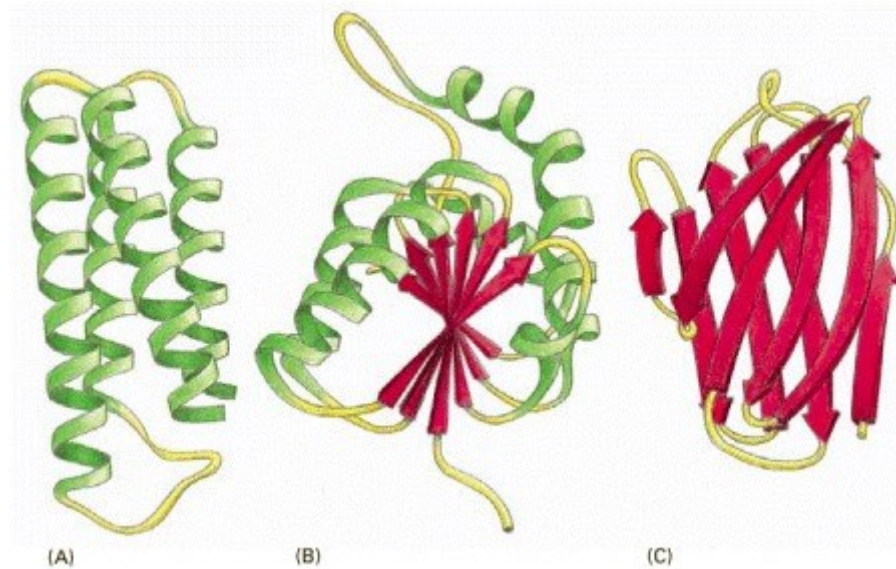
# The Protein Domain Is a Fundamental Unit of Organization



A protein formed from four domains

In the Src protein shown, two of the domains form a protein kinase enzyme, while the SH2 and SH3 domains perform regulatory functions. (A) A ribbon model, with ATP substrate in red. (B) A spacing-filling model, with ATP substrate in red. Note that the site that binds ATP is positioned at the interface of the two domains that form the kinase.

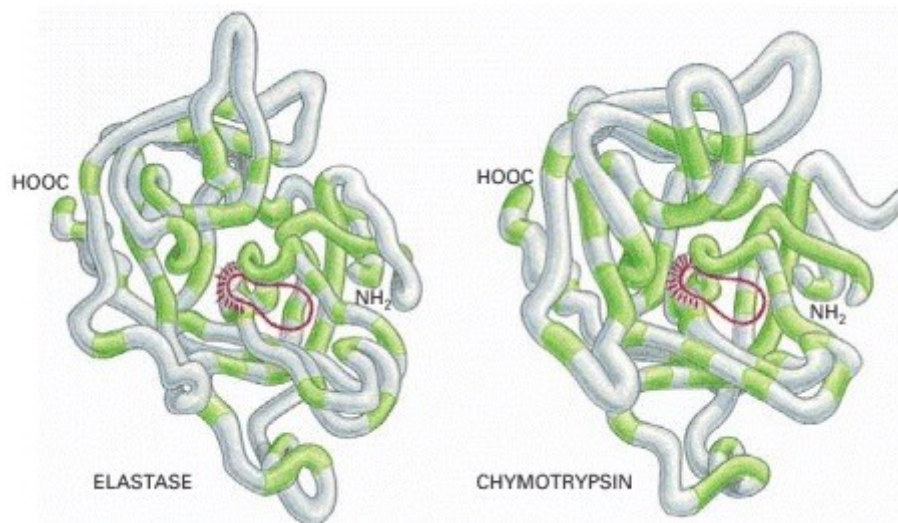
# The Protein Domain Is a Fundamental Unit of Organization



Ribbon models of three different protein domains

(A) Cytochrome b562, a single-domain protein involved in electron transport in mitochondria. This protein is composed almost entirely of  $\alpha$  helices. (B) The NAD-binding domain of the enzyme lactic dehydrogenase, which is composed of a mixture of  $\alpha$  helices and  $\beta$  sheets. (C) The variable domain of an immunoglobulin (antibody) light chain, composed of a sandwich of two  $\beta$  sheets. In these examples, the  $\alpha$  helices are shown in green, while strands organized as  $\beta$  sheets are denoted by red arrows.

# Proteins Can Be Classified into Many Families

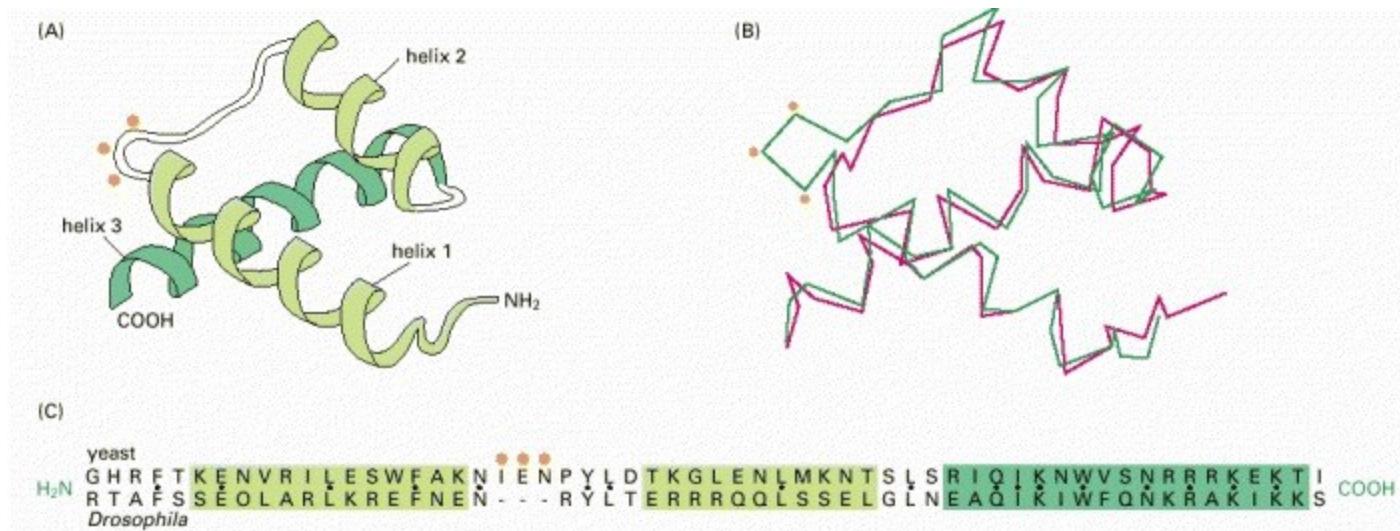


The conformations of two serine proteases compared

The backbone conformations of elastase and chymotrypsin. Although only those amino acids in the polypeptide chain shaded in green are the same in the two proteins, the two conformations are very similar nearly everywhere. The active site of each enzyme is circled in red; this is where the peptide bonds of the proteins that serve as substrates are bound and cleaved by hydrolysis. The serine proteases derive their name from the amino acid serine, whose side chain is part of the active site of each enzyme and directly participates in the cleavage reaction.



# Proteins Can Be Classified into Many Families

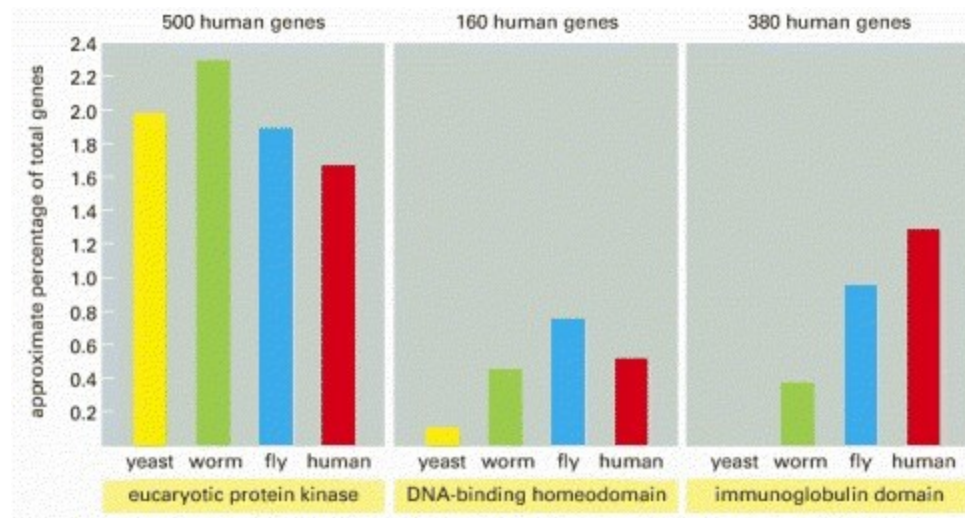


A comparison of a class of DNA-binding domains, called homeodomains, in a pair of proteins from two organisms separated by more than a billion years of evolution

(A) A ribbon model of the structure common to both proteins. (B) A trace of the  $\alpha$ -carbon positions. The three-dimensional structures shown were determined by x-ray crystallography for the yeast  $\alpha 2$  protein (green) and the *Drosophila* engrailed protein (red). (C) A comparison of amino acid sequences for the region of the proteins shown in (A) and (B). Black dots mark sites with identical amino acids. Orange dots indicate the position of a three amino acid insert in the  $\alpha 2$  protein.



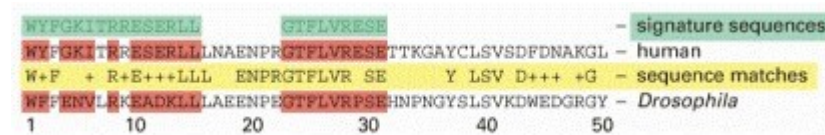
# Proteins Can Be Classified into Many Families



Percentage of total genes containing one or more copies of the indicated protein domain, as derived from complete genome sequences

Note that one of the three domains selected, the immunoglobulin domain, has been a relatively late addition, and its relative abundance has increased in the vertebrate lineage. The estimates of human gene numbers are approximate.

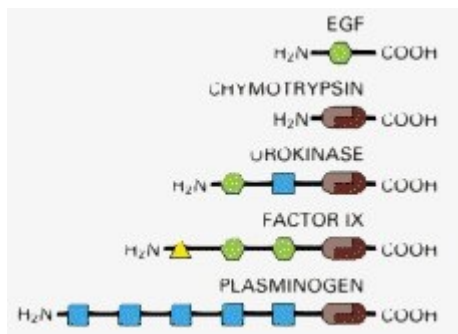
# Sequence Homology Searches Can Identify Close Relatives



The use of short signature sequences to find homologous protein domains

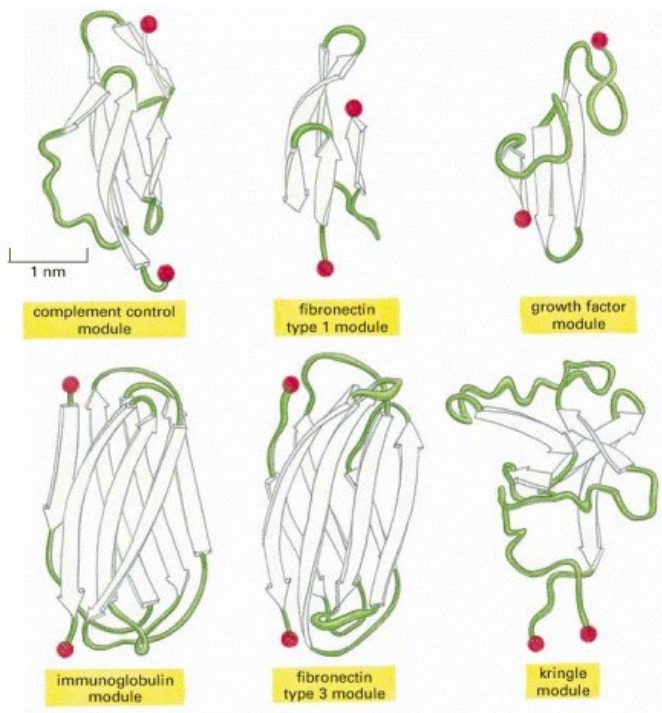
The two short sequences of 15 and 9 amino acids shown (green) can be used to search large databases for a protein domain that is found in many proteins, the SH2 domain. Here, the first 50 amino acids of the SH2 domain of 100 amino acids is compared for the human and Drosophila Src protein. In the computer-generated sequence comparison (yellow row), exact matches between the human and Drosophila proteins are noted by the one-letter abbreviation for the amino acid; the positions with a similar but nonidentical amino acid are denoted by +, and nonmatches are blank. In this diagram, wherever one or both proteins contain an exact match to a position in the green sequences, both aligned sequences are colored red.

# Some Protein Domains, Called Modules, Form Parts of Many Different Proteins



An extensive shuffling of blocks of protein sequence (protein domains) has occurred during protein evolution. Those portions of a protein denoted by the same shape and color in this diagram are evolutionarily related. Serine proteases like chymotrypsin are formed from two domains (brown). In the three other proteases shown, which are highly regulated and more specialized, these two protease domains are connected to one or more domains homologous to domains found in epidermal growth factor (EGF; green), to a calcium-binding protein (yellow), or to a “kringle” domain (blue) that contains three internal disulfide bridges.

# Some Protein Domains, Called Modules, Form Parts of Many Different Proteins



The three-dimensional structures of some protein modules

In these ribbon diagrams,  $\beta$ -sheet strands are shown as arrows, and the N- and C-termini are indicated by red spheres

# Review Questions

- Explain structural organization of peptides.
- What are the main characteristics of each level of organization of peptide molecules?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 4. Biological Databases

### ☐ Topic 2. Primary and Secondary Databases of Protein. 3D structure protein databases.

#### ☐ Lesson 1. Primary and Secondary Databases of Protein



ERASMUS+



# PRIMARY AND SECONDARY DATABASES OF PROTEIN

GMR

*Review*

## Bioinformatics: an overview and its applications

W.J.S. Dias<sup>a</sup> and F. Canduri<sup>b</sup>

<sup>a</sup>Departamento de Genética e Evolução,  
Universidade Federal de São Carlos, São Carlos, SP, Brasil

<sup>b</sup>Departamento de Química e Física Molecular,  
Instituto de Química de São Carlos, Universidade de São Paulo,  
São Carlos, SP, Brasil

Corresponding author: F. Canduri  
E-mail: fcanduri@iqsc.usp.br

Genet. Mol. Res. 16 (1): gmr16019645

Received February 16, 2017

Accepted March 2, 2017

Published March 15, 2017

DOI <http://dx.doi.org/10.4238/gmr16019645>

Copyright © 2017 The Authors. This is an open-access article distributed under the terms of  
the Creative Commons Attribution ShareAlike (CC BY-SA) 4.0 License.

**ABSTRACT.** Technological advancements in recent years have promoted a marked progress in understanding the genetic basis of phenotypes. In line with these advances, genomics has changed the paradigm of biological questions in full genome-wide scale (genome-wide), revealing an explosion of data and opening up many possibilities. On the other hand, the vast amount of information that has been generated points the challenges that must be overcome for storage (Moore's law) and processing of biological information. In this context, bioinformatics and computational biology have sought to overcome such challenges. This review presents an overview of bioinformatics and its use in the analysis of biological data, exploring approaches, emerging methodologies, and tools that can give biological meaning to the data generated.

**Key words:** Data analysis; Databases; Genomics; Systems biology

Genetics and Molecular Research 16 (1): gmr16019645

[https://www.geneticsmr.com/sites/default/files/articles/year2017/vol16-1/pdf/gmr-16-01-gmr.16019645\\_0.pdf](https://www.geneticsmr.com/sites/default/files/articles/year2017/vol16-1/pdf/gmr-16-01-gmr.16019645_0.pdf)

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



# Contents

- Introduction
- Biological Databases
- Uses of biological Databases
- Primary databases
- Secondary databases
- Composite Databases (Hybrid databases and families of databases)

# Introduction

As the number of published sequences increased, the workflow changed: by opening discussions with publishers of the scientific literature, the organisations behind these databases convinced publishers to request that researchers submit their sequences to one of the public databases before submitting the paper. In return, authors gained an accession number that could then be cited in the paper. This model has now been followed by many providers of public biological data.

The other important aspect of these collaborations is that their participants exchange data and/or assign workload in such a way as to avoid duplication of effort whilst ensuring that data are annotated and made available in a consistent way.

**Biological databases store and organize biological data for easy retrieval of information. These centralized resources contain DNA and protein sequences, and their associated information.**

# Biological Databases

- These are the databases consisting of biological data like protein sequencing, molecular structure, DNA sequences, etc in an organized form.
- Several computer tools are there to manipulate the biological data like an update, delete, insert, etc. Scientists, researchers from all over the world enter their experiment data and results in a biological database so that it is available to a wider audience.
- Biological databases are free to use and contain a huge collection of a variety of biological data.

# Uses of biological Databases

- It helps the researchers to study the available data and form a new thesis, anti-virus, helpful bacteria, medicines, etc.
- It helps scientists to understand the concepts of biological phenomena.
- The database acts as a storage of information.
- It helps remove the redundancy of data

## Some examples of global collaborations established to manage the public record of different biological data types

Data type	Collaboration
Nucleotide sequences	<a href="#"><u>International Sequence Database Collaboration</u></a>
Protein sequences	<a href="#"><u>UniProt Consortium</u></a>
Macromolecular structures	<a href="#"><u>Worldwide Protein Data Bank</u></a>
Molecular interactions	<a href="#"><u>The International Molecular Exchange Consortium</u></a>
Protein identifications	<a href="#"><u>The ProteomeXchange Consortium</u></a>
Genomic and clinical data	<a href="#"><u>Global Alliance for Genomics and Health</u></a>

# Primary databases

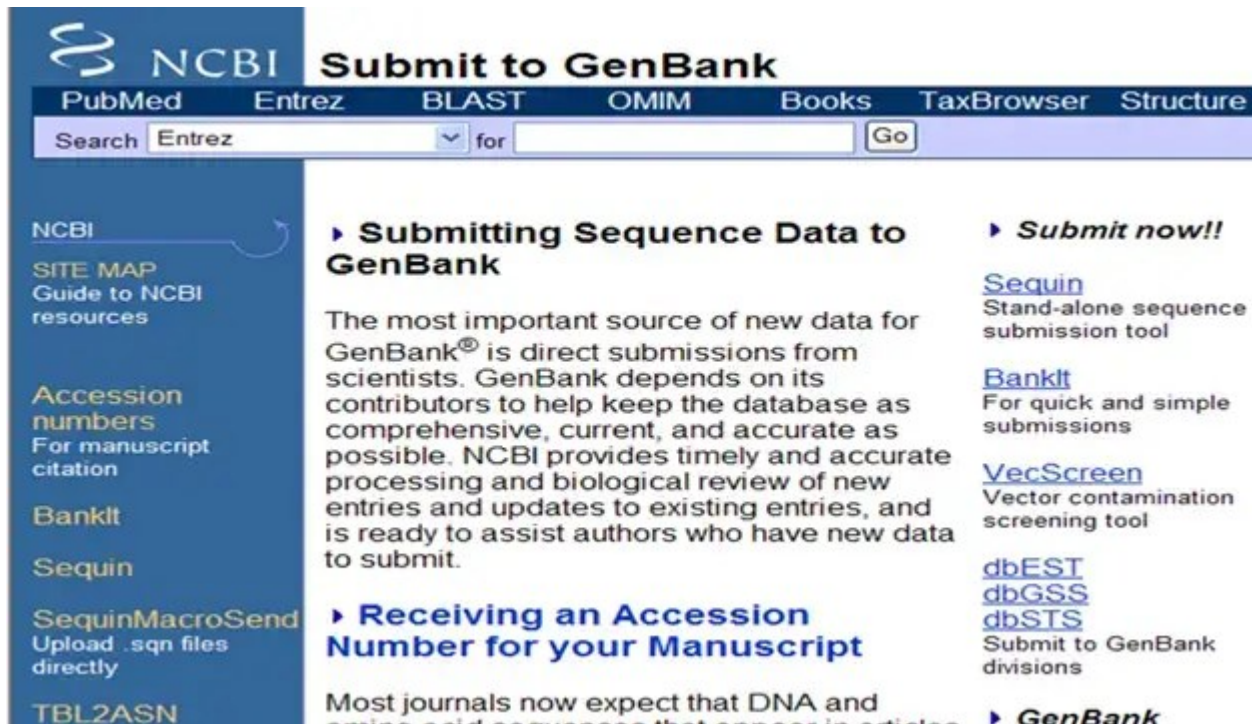
In bioinformatics, and indeed in other data intensive research fields, databases are often categorised as primary or secondary. Primary databases are populated with experimentally derived data such as nucleotide sequence, protein sequence or macromolecular structure. Experimental results are submitted directly into the database by researchers, and the data are essentially archival in nature. Once given a database accession number, the data in primary databases are never changed: they form part of the scientific record.



# Primary databases

- In bioinformatics, and indeed in other data intensive research fields, databases are often categorised as primary or secondary. Primary databases are populated with experimentally derived data such as nucleotide sequence, protein sequence or macromolecular structure. Experimental results are submitted directly into the database by researchers, and the data are essentially archival in nature. Once given a database accession number, the data in primary databases are never changed: they form part of the scientific record.

# Primary databases



The screenshot shows the NCBI 'Submit to GenBank' page. At the top, there's a navigation bar with links to PubMed, Entrez, BLAST, OMIM, Books, TaxBrowser, and Structure. Below this is a search bar with 'Entrez' selected and a 'Go' button. The main content area is divided into two columns. The left column contains links to 'NCBI SITE MAP', 'Accession numbers', 'BankIt', 'Sequin', 'SequinMacroSend', and 'TBL2ASN'. The right column features a large heading 'Submitting Sequence Data to GenBank' followed by a paragraph explaining the importance of direct submissions. To the right of this paragraph are links for 'Sequin', 'BankIt', 'VecScreen', 'dbEST', 'dbGSS', and 'dbSTS'. At the bottom of the right column, there's a heading 'Receiving an Accession Number for your Manuscript' and a paragraph about journal expectations. A 'GenBank' link is also present at the bottom right.

**NCBI Submit to GenBank**

PubMed Entrez BLAST OMIM Books TaxBrowser Structure

Search Entrez for Go

**NCBI**  
SITE MAP  
Guide to NCBI resources

**Accession numbers**  
For manuscript citation

**BankIt**

**Sequin**

**SequinMacroSend**  
Upload .sqn files directly

**TBL2ASN**

► **Submitting Sequence Data to GenBank**

The most important source of new data for GenBank® is direct submissions from scientists. GenBank depends on its contributors to help keep the database as comprehensive, current, and accurate as possible. NCBI provides timely and accurate processing and biological review of new entries and updates to existing entries, and is ready to assist authors who have new data to submit.

► **Submitting Sequence Data to GenBank**

► **Submit now!!**

[Sequin](#)  
Stand-alone sequence submission tool

[BankIt](#)  
For quick and simple submissions

[VecScreen](#)  
Vector contamination screening tool

[dbEST](#)  
[dbGSS](#)  
[dbSTS](#)  
Submit to GenBank divisions

► **Receiving an Accession Number for your Manuscript**

Most journals now expect that DNA and amino acid sequences that are submitted to GenBank

► **GenBank**

# Secondary databases

Secondary databases comprise data derived from the results of analyzing primary data. They are often referred to as curated databases but this is a bit of a misnomer because primary databases are also curated to ensure that the data in them is consistent and accurate.

Secondary databases often draw upon information from numerous sources, including other databases (primary and secondary), controlled vocabularies (see later section) and the scientific literature. They are highly curated, often using a complex combination of computational algorithms and manual analysis and interpretation to derive new knowledge from the public record of science.

Secondary databases have become the molecular biologist's reference library over the past decade or so, providing a wealth of (often daunting) information on just about any gene or gene product that has been investigated by the research community. The potential for mining this information to make new discoveries is vast. It's our job in this course to reduce your activation energy to make more of these resources for your research.

# Essential aspects of primary and secondary databases

	Primary database	Secondary database
<b>Synonyms</b>	Archival database	Curated database; knowledgebase
<b>Source of data</b>	Direct submission of experimentally-derived data from researchers	Results of analysis, literature research and interpretation, often of data in primary databases
<b>Examples</b>	<a href="#">ENA</a> , <a href="#">GenBank</a> and <a href="#">DDBJ</a> (nucleotide sequence) <a href="#">ArrayExpress</a> and <a href="#">GEO</a> (functional genomics data) <a href="#">Protein Data Bank</a> (PDB; coordinates of three-dimensional macromolecular structures)	<a href="#">InterPro</a> (protein families, motifs and domains) <a href="#">UniProt Knowledgebase</a> (sequence and functional information on proteins) <a href="#">Ensembl</a> (variation, function, regulation and more layered onto whole genome sequences)

# SWISS-PROT

- SWISS-PROT is a well-known and widely used secondary database of protein sequences that provides detailed annotation, including information on structure, function, and protein family assignment.
- The sequence data is primarily derived from the TrEMBL database, which stores translated nucleic acid sequences.
- SWISS-PROT stands out from other protein databases for its detailed annotations, minimal redundancy, and integration with other databases.
- Annotations in SWISS-PROT provide detailed information on protein function, post-translational modifications, domains and sites, secondary and quaternary structure, similarities to other proteins, diseases associated with deficiencies in the protein, sequence conflicts, and variants.
- Swiss-Prot is popular for its low redundancy and high level of integration with other databases.



# PROSITE

- ProSite is a database of protein families, domains, and functional sites that contains manually curated information on amino acid patterns and profiles of proteins.
- It is a secondary protein database that provides tools for the analysis of protein sequences and the identification of motifs.
- The database contains a large collection of signature patterns or profiles that hold biological importance. Each signature is associated with important biological information such as protein family, domain, or functional site.
- ProSite uses two types of signatures, patterns and generalized profiles, to identify conserved regions.
- These signatures can be used to predict the function and structure of proteins and help in the annotation of new protein sequences.

# Pfam

- Pfam is another secondary database of protein families and domains that are represented by multiple sequence alignments, profile hidden Markov models (HMMs), and annotations.
- The database is accessible online and is used by researchers worldwide for a variety of applications, including genome annotation, protein classification, and protein structure prediction.
- Pfam has two components. Pfam-A stores manually curated high-quality entries. Pfam-B stores automatically generated lower-quality entries.
- Pfam provides a platform for the analysis of protein sequence data, which allows researchers to search for related proteins in the database based on the presence of specific protein domains.



# PRINTS

- PRINTS database contains protein family fingerprints which are groups of motifs.
- PRINTS is one of several widely-used pattern databases, including PROSITE, BLOCKS, and Pfam, each with different strengths and weaknesses.
- PRINTS uses a fingerprinting method that detects distant relatives of large and highly divergent protein superfamilies by exploiting conserved regions within sequence alignments.

# Applications of Secondary Databases

- Secondary databases can be used to predict the structure and function of proteins by identifying homologous proteins with known structures.
- Secondary databases contain functional annotation information which helps to better understand the roles of proteins in different organisms.
- Secondary databases also help to identify conserved regions within a sequence, which can help to identify important functional domains and motifs.
- Secondary databases also help in evolutionary analysis by comparing protein sequences across different species to study the evolution of proteins.
- Secondary databases can also be used to identify potential drug targets by analyzing protein families and identifying conserved motifs that are essential for protein function.

# Composite Databases (Hybrid databases and families of databases)

- The data entered in these types of databases are first compared and then filtered based on desired criteria.
- The initial data are taken from the primary database, and then they are merged together based on certain conditions.
- It helps in searching sequences rapidly. Composite Databases contain non-redundant data.

# Hybrid databases and families of databases

Many data resources have both primary and secondary characteristics. For example, UniProt accepts primary sequences derived from peptide sequencing experiments. However, UniProt also infers peptide sequences from genomic information, and it provides a wealth of additional information, some derived from automated annotation (TrEMBL), and even more from careful manual analysis (SwissProt).

Some databases have different ‘branches’ for primary and secondary data. A good example of this is the ArrayExpress data collection in BioStudies: ArrayExpress contains experimentally-derived functional genomics data whereas the Expression Atlas uses a subset of high-quality data from the ArrayExpress collection to derive knowledge about gene expression patterns under different conditions.

To learn more about some EMBL-EBI databases, try the online tutorial [A journey through bioinformatics: Explore resources from EMBL-EBI](#).

# Review Questions

- Explain the term biological database.
- Explain the difference between primary and secondary databases.
- Give some examples of two types of databases.



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 4. Biological Databases

### ☐ Topic 2. Primary and Secondary Databases of Protein. 3D structure protein databases.

#### ☐ Lesson 2. 3D structure protein databases



ERASMUS+



# 3D STRUCTURE PROTEIN DATABASES

- Laskowski, R. A. (2011). Protein Structure Databases. *Molecular Biotechnology*, 48(2), 183–198.
- Berman, H. M. (January 2008). "The Protein Data Bank: a historical perspective" (PDF). *Acta Crystallographica Section A*. A64 (1): 88–95.



# Contents

- Introduction
- Terminology
- Atlases
- Examples of databases

# Introduction

Back in 1971, when the Protein Data Bank (PDB) was founded, there were only seven experimentally determined protein 3D structures. The data for each, including the proteins' atomic coordinates, were stored in simple, fixed-format text files. Any structural analysis of these proteins usually required access to bulky and expensive graphics computers.

Since then, the number of solved structures has increased ten thousand-fold. The internet revolution has helped by making access to, and display of, protein structural data vastly easier and providing a greater level of information content.

# Terminology

‘**Protein structure**’ is a term that tends to be somewhat loosely used.

A preferable term is ‘**model**’, as the 3D structures of large molecules, such as proteins, are models of the protein atoms’ locations (i.e. their x-, y-, z-coordinates), their chemical types (e.g. oxygen, nitrogen) and other parameters.

# Atlases

At the simplest level are the sites that provide ‘atlas’ pages—one for every PDB entry—each containing general information obtained from the relevant PDB file. There are usually graphical representations of the structural model together with links that provide interactive 3D visualizations using java-based, or other, viewers.

# Atlases

Server	Location	URL
JenaLib	Fritz Lipmann Institute, Jena, Germany	<a href="http://www.imb-jena.de/IMAGE.html">http://www.imb-jena.de/IMAGE.html</a>
MMDB	NCBI, USA	<a href="http://www.ncbi.nlm.nih.gov/Structure/MMDB/mmdb.shtml">http://www.ncbi.nlm.nih.gov/Structure/MMDB/mmdb.shtml</a>
PDBe	EBI, Cambridge, UK	<a href="http://www.ebi.ac.uk/pdbe">http://www.ebi.ac.uk/pdbe</a>
OCA	Weizmann Institute, Israel	<a href="http://bip.weizmann.ac.il/oca-docs/oca-home.html">http://bip.weizmann.ac.il/oca-docs/oca-home.html</a>
PDBj	Osaka University, Japan	<a href="http://www.pdbj.org">http://www.pdbj.org</a>
PDBsum	EBI, Cambridge, UK	<a href="http://www.ebi.ac.uk/pdbsum">http://www.ebi.ac.uk/pdbsum</a>
RCSB	Rutgers and San Diego, USA	<a href="http://www.rcsb.org/pdb">http://www.rcsb.org/pdb</a>

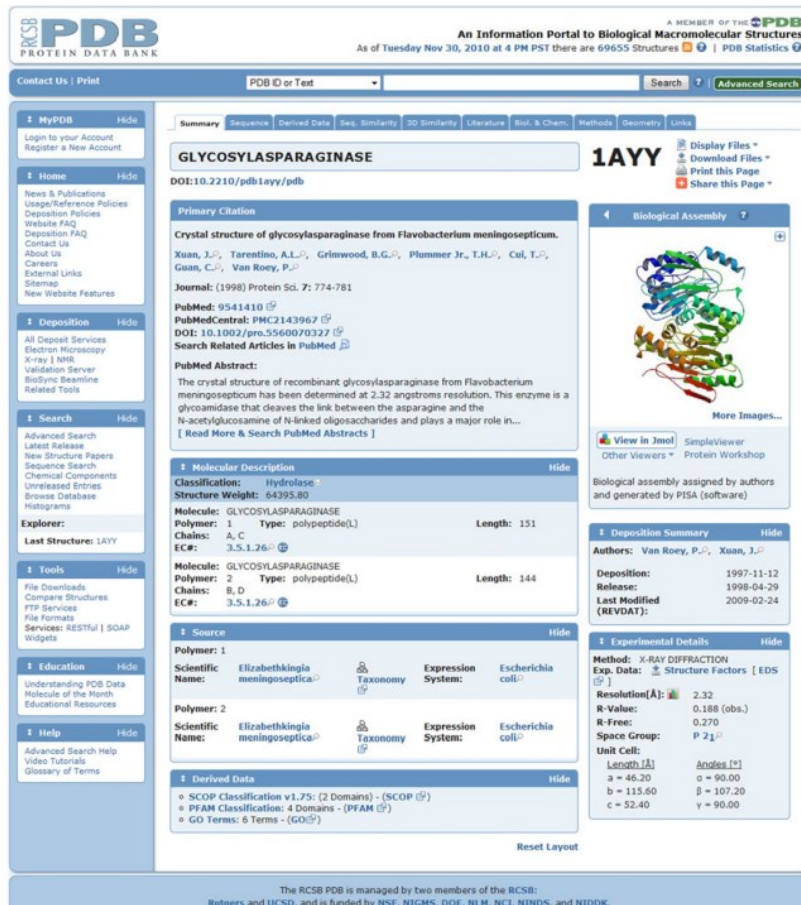
# The RCSB PDB

The RCSB's web site (<http://www.rcsb.org/pdb>) is a very rich source of information about each PDB entry and can be a little overwhelming for novices. Hence there are various tutorials, including a narrated one using Flash, to help users get started.



# The RCSB PDB - atlas page

RCSB atlas page for PDB entry 1ayy, a glycosylasparaginase showing the summary information for this structural model determined by X-ray crystallography at 2.32 Å resolution



The screenshot shows the RCSB PDB atlas page for entry 1ayy. The page is titled "GLYCOSYLASPARAGINASE" and "1AYY". It includes a summary of the entry, a primary citation, a molecular description, a deposition summary, and experimental details. The primary citation is: Xuan, J., Tarentino, A.L., Grisewood, B.G., Plummer Jr., T.H., Cai, T., Gao, C., Van Roey, P. (1998) Protein Sci. 7: 774-781. The molecular description indicates that the structure was determined at 2.32 Å resolution. The deposition summary shows that the structure was deposited on 1997-11-12 and released on 1998-04-29. The experimental details section shows that the method used was X-RAY DIFFRACTION, with a resolution of 2.32 Å.

At the top of each PDB entry's summary page is shown the primary literature citation for the entry, if there is one. The citation usually refers to the authors' description of the structure determination experiment, analysis performed on the resultant model and its biological significance. Next comes a description of the molecules making up the structure that was solved: protein chains, DNA fragments, ligand molecules, etc. Also given are various details of the experiment, including the organism from which the protein came and how it was expressed. Additional annotations, with appropriate links to the relevant databases are also given, including: the SCOP and CATH fold classifications, constituent Pfam domains, and Gene Ontology (GO) functional annotations. Text in bold blue initiates a search for all other PDB entries having that text in common (e.g. other entries with the same author name, or species, or protein classification, etc.).

ERASMUS+

# The RCSB PDB - Sequence details page

Summary Sequence Annotations Seq. Similarity 3D Similarity Literature Biol. & Chem. Methods Geometry Links

**GLYCOSYLASPARAGINASE** **1AYY** [Display Files](#) [Download Files](#) [Share this Page](#)

Sequence / Structure Details

**Redundancy Reduction and Sequence Clustering**  
View the clustering results for 1AYY.

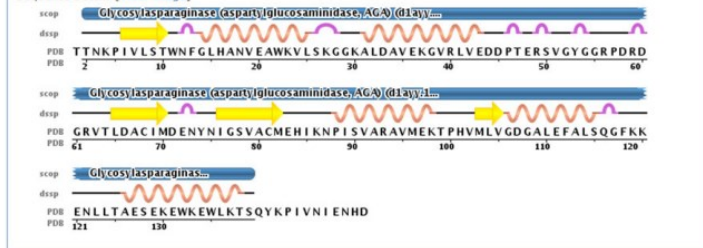
**Sequence Display** [i](#)  
The structure **1AYY** has in total **4** chains. Out of these **2** are sequence-unique.  
Currently viewing **unique chains** only. [\[show all chains\]](#) [\[Show 3D in Jmol\]](#)

**Chain Display**

Chain A (polymer 1) [\[help\]](#) [\[fasta\]](#) [\[sequence & DSSP\]](#)

Description GLYCOSYLASPARAGINASE  
Identical chains C  
[\[show all chains\]](#)  
Chain Type polypeptide(L)  
UniProtKB reference Q47898  
Length 151 residues  
scop domain assignment **1.10.1.1** Glycosylasparaginase (aspartylglucosaminidase, AGA): 138 residues [\[hide\]](#) [\[reference\]](#)  
dssp secondary structure 39% helical (5 helices; 59 residues)  
[\[hide\]](#) [\[reference\]](#)  
More annotations  
Select

Currently displayed: SEQRES sequence. [\[display external \(UniProtKB\) sequence\]](#)  
Sequence Details [\[View image\]](#)

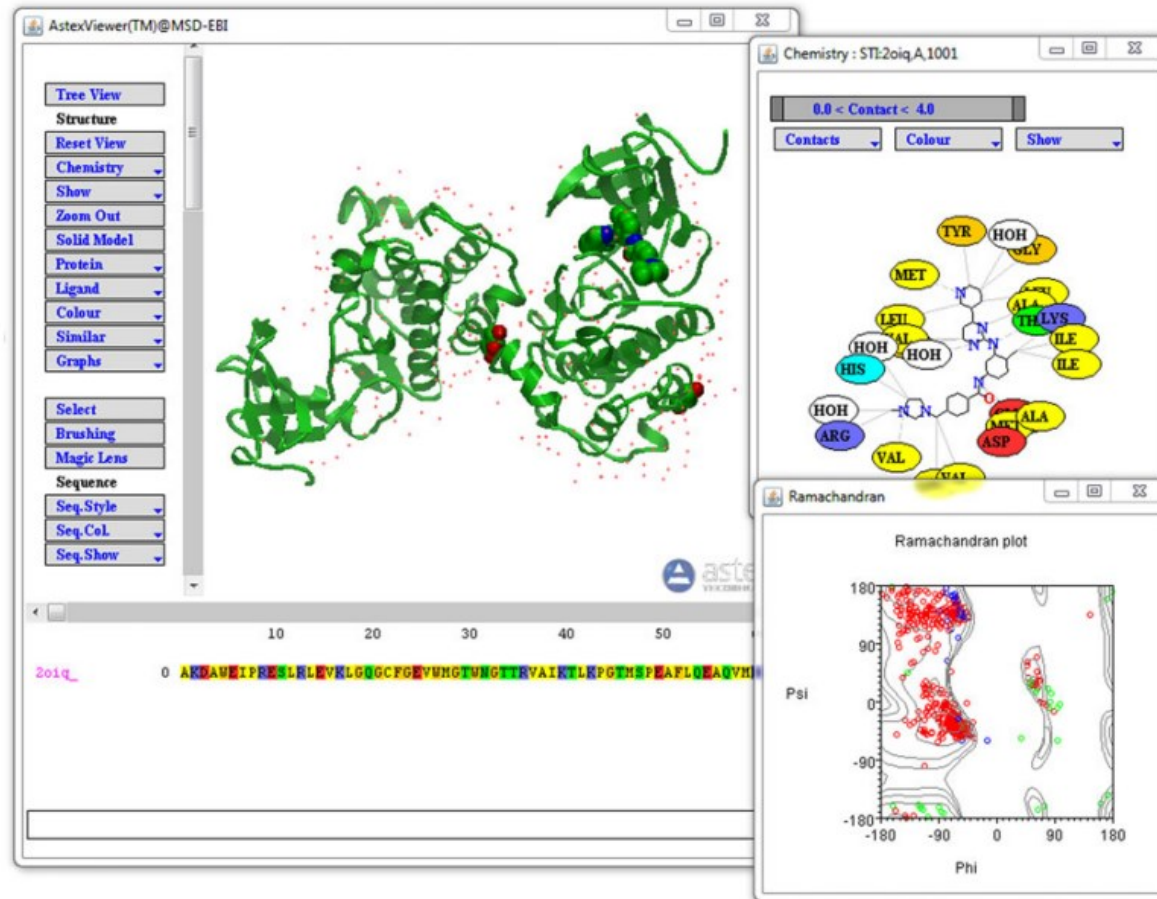


The Sequence details page which includes a schematic diagram of the protein's secondary structure (showing  $\alpha$ - and  $\beta$ -helices,  $\beta$ -sheets and turns), together with any SCOP structural domains. The 3D Similarity page allows you to find other protein structural models that are similar in overall fold to the current PDB entry.

# The PDBe

The PDBe, formerly known as the Macromolecular Structure Database (MSD), is the European branch of the wwPDB. Its site provides an extensive set of search and analysis tools that allow one to explore and mine the structural data in the PDB (<http://www.ebi.ac.uk/pdbe>).

The atlas pages for each entry show the usual summary information describing the structure and the experimental details used to obtain it. Additional pages provide information on the protein's Primary, Secondary, Tertiary and Quaternary structure (i.e. the probable biological unit, as predicted by PISA), Experimental method, Taxonomy information, Citation, Ligand and Visualization.



Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



# OCA

OCA's main difference from the other atlases is its linkage between proteins and the diseases associated with them (<http://bip.weizmann.ac.il/oqa-docs/oqa-home.html>). Its search form has a few novel search options including gene name, function, disease and membrane orientation (for membrane-spanning proteins).



OCA Browser

## some interesting entries

1ehz (structure with most ligands)  
1ofd (structure with the longest chain)  
1ir1 (multi-specie gene)  
1a1u (rich functional annotation)  
4sbv (representative and supersedes)  
1fba (gene Drosophila melanogaster)  
1adn (gene Escherichia coli)  
1bte (gene Mus musculus)  
1xum (G protein-coupled receptors)  
1bgx (structure with most domains)

OCA®, a browser-database for protein structure/function.

Oca is the spanish word for **goose**.

(As Merriam-Webster says "goose: any of numerous large waterfowl (family Anatidae) that are intermediate between the swans and ducks and have long necks, feathered lores, and reticulate tarsi").

## the largest ligands in PDB

**BLB** C55 H85 N20 O21 S2  
**PL1** C48 H68 N17 O12 CO1 S2 1+  
**VAX** C73 H91 N10 O26 CL2 3+  
**VAN** C66 H77 N9 O24 CL2 2+  
**THX** C50 H53 N10 O12 P1  
**NTP** C36 H60 O55 S9  
**CDL** C81 H156 O17 P2 2-  
**NCG** C45 H53 N4 O18 S1  
**CAS** C33 H47 N9 O17 BR1 P3 S1  
**BLM** C55 H85 N17 O21 S3

**tips**, suggestions and answers on OCA® are available in the [FAQ](#) document.

**reports** generated by OCA® weekly on PDB data:

- Number Biomolecules Distribution
- File size Distribution
- Distribution of PDB Records Names
- ALTLOC in PDB structures
- MODRES in PDB structures

**links** OCA® provides rich content annotation on structure and function, generating dynamic links to several [external sources](#).

**sources** OCA® integrates data from several [sources](#) that are of free access for Universities. If you plan to use OCA® in a commercial environment or for profit, please be sure you have the proper clearance from OCA® and all the OCA® data [sources](#).

**internals** and working details on the [engines](#) that make OCA® unique.

**mirrors** OCA® is being mirrored worldwide at several [mirror sites](#).

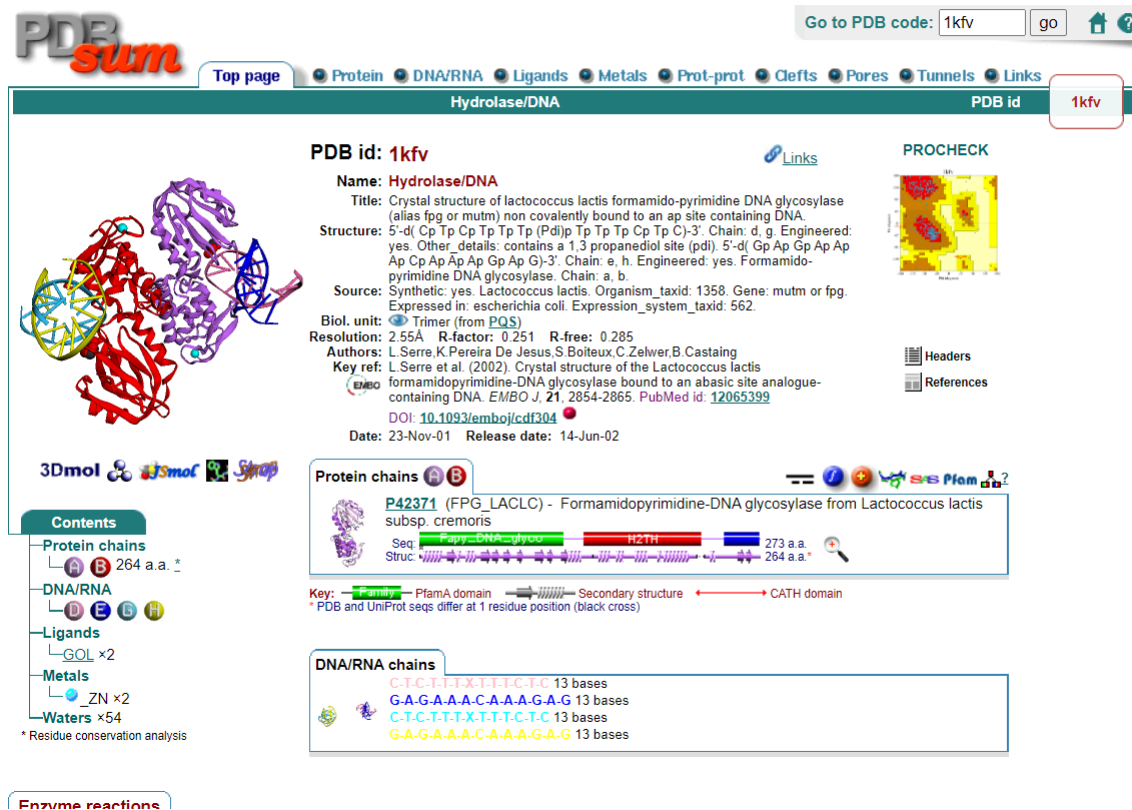
**download** If you have interest in running your local copy of OCA®, here is the information on how to become a [mirror](#).

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# PDBsum

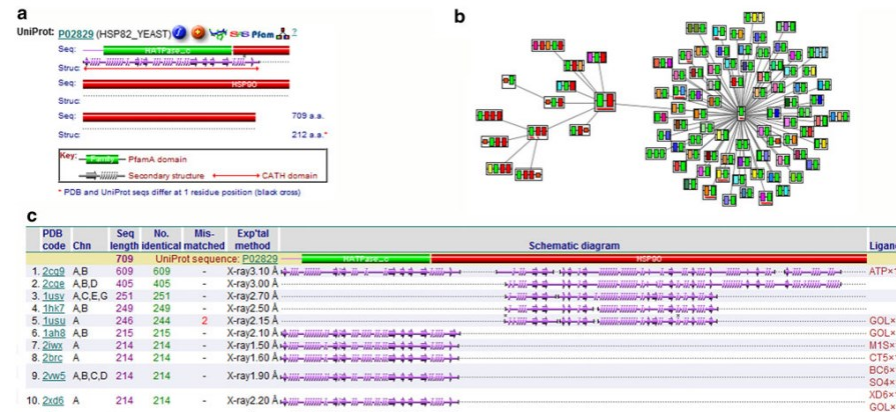
## PDBsum entry 1kfv



This aims to be more pictorial than the other sites, illustrating many of its structural analyses by schematic diagrams rather than as tables of numbers (<https://www.ebi.ac.uk/thornton-srv/databases/pdbsum/>). Also, it allows users to upload their own PDB files and get a set of password-protected PDBsum pages generated for them.

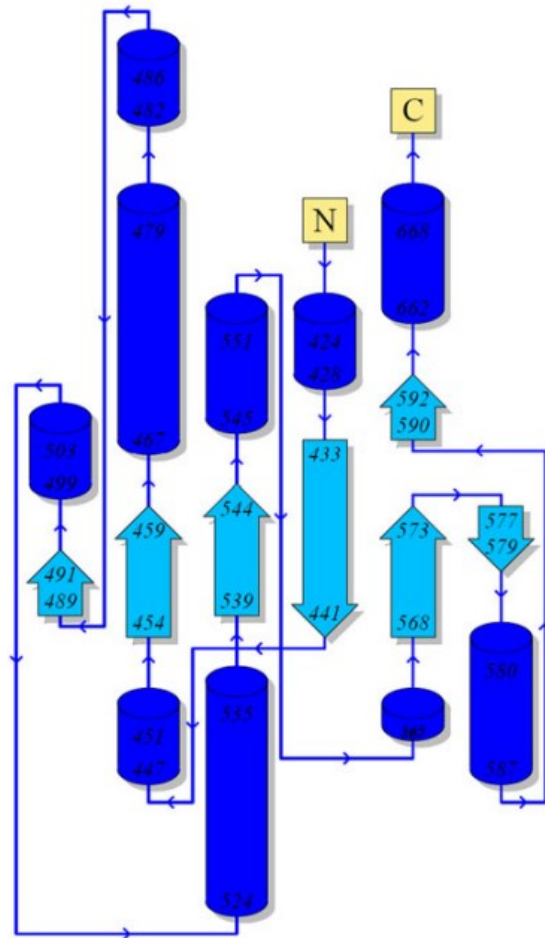
# Pfam Domain Diagrams and Domain Architecture Networks

Each entry's summary page has a few useful features not found in the other atlas sites. One of these is a clickable schematic diagram showing how much of the full-length protein sequence is actually represented by the 3D structural model.



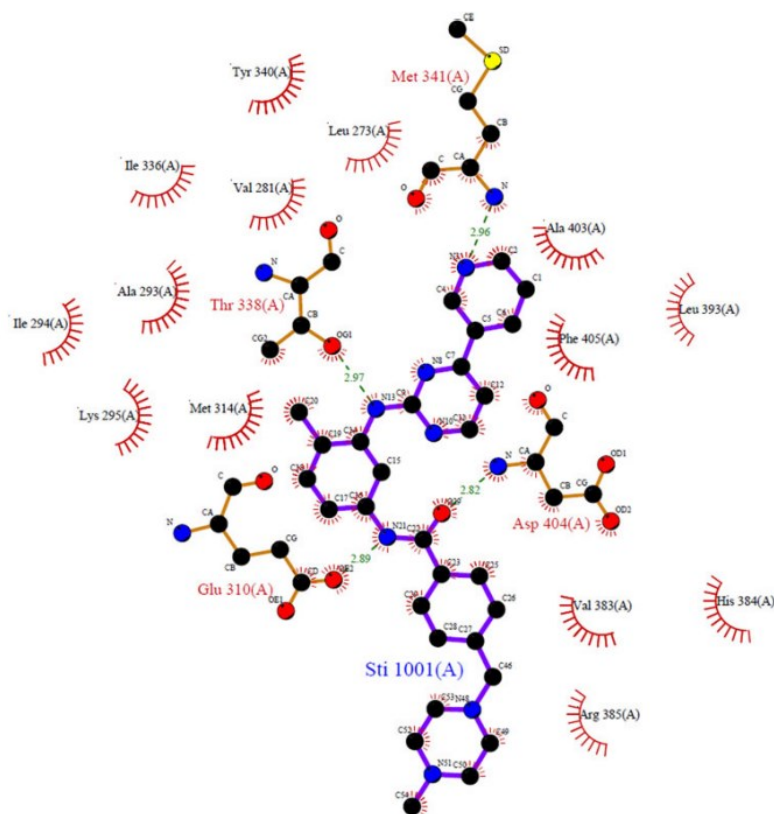


# Secondary Structure and Topology Diagrams



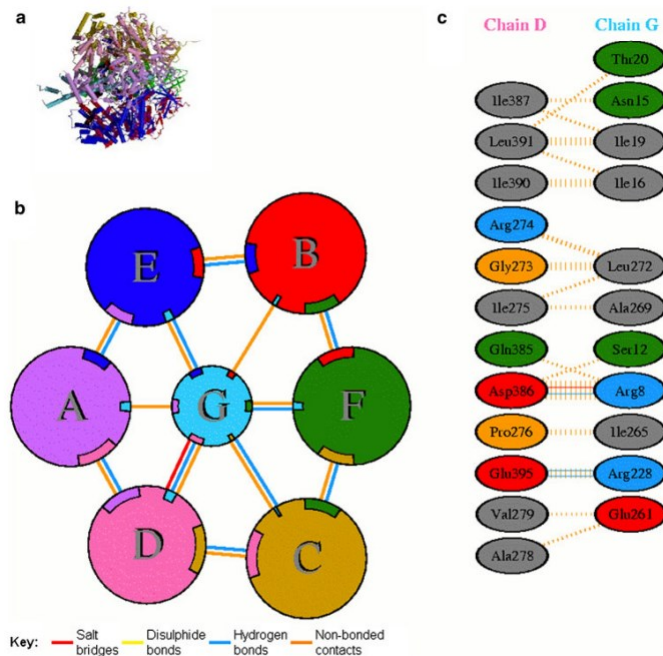
A topology diagram taken from PDBsum for the second domain of chain A in PDB entry 3ib0: a bovine lactoferrin. The diagram illustrates how the  $\beta$ -strands, represented by the block arrows, join up, side-by-side, to form the domain's central  $\beta$ -sheet. The diagram also shows the relative locations of the  $\alpha$ -helices, here represented by cylinders. The small arrow indicates the directionality of the protein chain, from the N- to the C-terminus. The numbers within the secondary structural elements correspond to the residue numbering given in the PDB file.

# Intermolecular Interactions



LIGPLOT for PDB entry 2oiq, tyrosine kinase c-Src, as given in PDBsum showing the interactions between the bound molecule imatinib (a drug, brand name gleevec) with the residues of the protein. Hydrogen bonds are represented by dashed lines. Residues that interact with the ligand via non-bonded contacts only are represented by the eyelashes.

# Intermolecular Interactions



Extracts from the protein–protein interaction diagrams in PDBsum for PDB entry 1cow, bovine mitochondrial F1-ATPase. **a** Thumbnail image of the 3D structural model which contains seven protein chains: three of ATPA1\_BOVIN (chains A, B and C), three of ATPB\_BOVIN (chains D, E and F) and a fragment of ATPG\_BOVIN (chain G). **b** Schematic diagram showing the interactions between the chains. The area of each circle is proportional to the surface area of the corresponding protein chain. The extent of the interface region on each chain is represented by a coloured wedge whose colour corresponds to the colour of the other chain and whose size signifies the interface surface area. **c** A schematic diagram showing the residue–residue interactions across one of the interfaces, namely that between chains D and G. Hydrogen bonds and salt bridges are shown as solid lines, while nonbonded contacts are represented by dashed lines (Color figure online)

# Homology model servers

Server	Location	URL
(i) Automatic homology modelling		
3D-JIGSAW	Imperial Cancer Research Fund, UK	<a href="http://www.bmm.icnet.uk/servers/3djigsaw">http://www.bmm.icnet.uk/servers/3djigsaw</a>
CPHmodels	Technical University of Denmark	<a href="http://www.cbs.dtu.dk/services/CPHmodels">http://www.cbs.dtu.dk/services/CPHmodels</a>
ESyPred3D	University of Namur, Belgium	<a href="http://www.fundp.ac.be/urbm/bioinfo/esypred">http://www.fundp.ac.be/urbm/bioinfo/esypred</a>
SWISS-MODEL	Biozentrum Basel, Switzerland	<a href="http://swissmodel.expasy.org">http://swissmodel.expasy.org</a>
(ii) Evaluation of modelling servers		
EVA	Columbia University, USA	<a href="http://cubic.bioc.columbia.edu/eva">http://cubic.bioc.columbia.edu/eva</a>
(iii) Pre-computed homology models		
SWISS-MODEL Repository	Biozentrum Basel, Switzerland	<a href="http://swissmodel.expasy.org/repository">http://swissmodel.expasy.org/repository</a>
ModBase	University of California San Francisco, USA	<a href="http://modbase.compbio.ucsf.edu">http://modbase.compbio.ucsf.edu</a>
PDB archive	wwPDB	<a href="ftp://ftp.wwpdb.org/pub/pdb/data/structures/models">ftp://ftp.wwpdb.org/pub/pdb/data/structures/models</a>

# Review Questions

- Explain the role of protein structure databases.
- Give some examples of protein structure databases.
- What are the main features of such kind of databases?



Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 4. Biological Databases

### ☐ Topic 3. Gene databases

#### ☐ Lesson 1. Types of genome databases

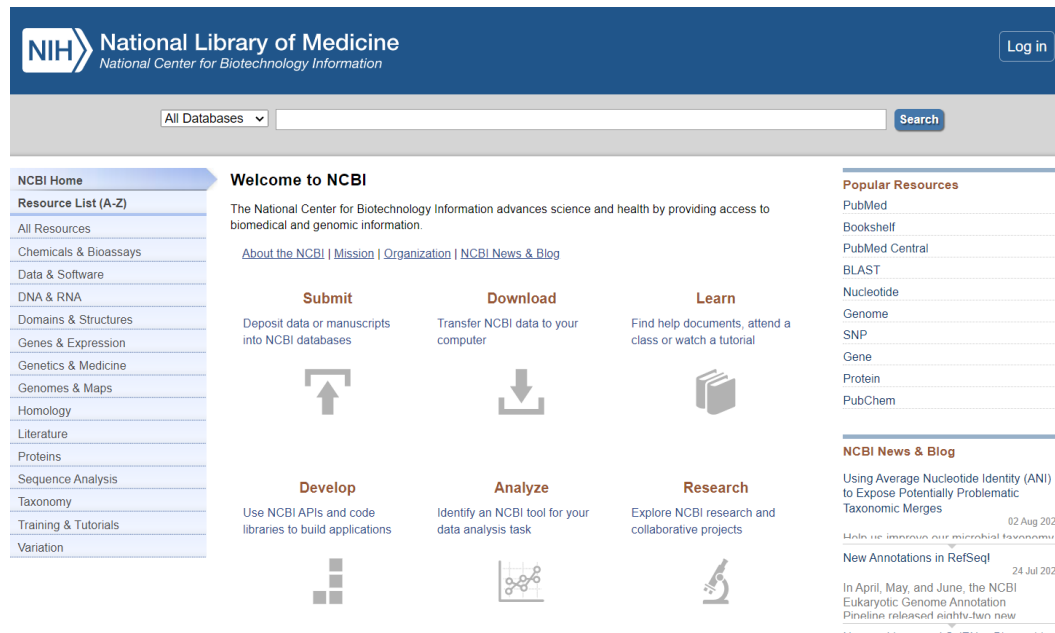


ERASMUS+



# TYPES OF GENOME DATABASES

- Carroll, M.L., Nguyen, S.V. and Batzer, M.A. (2001). Genome Databases. In eLS, (Ed.). <https://doi.org/10.1038/npg.els.0003027>



The screenshot shows the NCBI homepage with a dark blue header containing the NIH logo and the text "National Library of Medicine National Center for Biotechnology Information". A "Log in" button is in the top right. Below the header is a search bar with a dropdown menu set to "All Databases" and a "Search" button. The main content area is divided into three columns. The left column is a "Resource List (A-Z)" with links to various databases. The middle column is titled "Welcome to NCBI" and contains three main sections: "Submit" (Deposit data or manuscripts into NCBI databases), "Download" (Transfer NCBI data to your computer), and "Learn" (Find help documents, attend a class or watch a tutorial). Below these are "Develop" (Use NCBI APIs and code libraries to build applications), "Analyze" (Identify an NCBI tool for your data analysis task), and "Research" (Explore NCBI research and collaborative projects). The right column is titled "Popular Resources" and lists various databases like PubMed, Bookshelf, PubMed Central, BLAST, Nucleotide, Genome, SNP, Gene, Protein, and PubChem. Below this is a "NCBI News & Blog" section with recent news items.

# Contents

- Introduction
- Genome Data Management
- Genomic Databases
- Relational databases
- Object-oriented databases
- Commercial Databases
- Existing Genome Databases

# Introduction

A genome database can be described as a repository of DNA sequences from many different species of plants and animals. They are linked to supportive databases to aid in interpretation of the sequence data. Genome databases are designed and maintained in the electronic environment of one or several computers, using several operating systems, software applications, file transfer protocols and user interfaces.

Genome databases contain sequence data generated by molecular biologists, geneticists and others using techniques in the laboratory that enable determination of the individual nucleotide order of a complete DNA sequence.

# Genome Data Management

**Data Storage** – Storing large volumes of genome data requires a combination of scalable storage solutions and efficient data compression methods. Popular storage solutions include cloud storage, distributed file systems, and relational databases.

**Data Quality Control** – Quality control is essential for ensuring the accuracy and reliability of genome data. This includes checking for errors in sequencing, contamination, and data integrity.

**Data Analysis** – The complexity and diversity of genome data require a wide range of analytical tools and methods. These include alignment tools, variant calling, annotation, functional analysis, and visualization tools.

**Data Integration** – Integrating data from different sources and in different formats is a major challenge in genome data management. This requires the use of standard data formats, ontologies, and data integration tools.

**Data Security** – The sensitive nature of genome data requires strict security measures to protect the privacy of research participants and to comply with regulations. This includes data encryption, access controls, and data-sharing policies.

# Genomic Databases

Resource	URL
Bacteria Database	[ <a href="http://www.ncbi.nlm.nih.gov/PMGifs/Genomes/5833.html">http://www.ncbi.nlm.nih.gov/PMGifs/Genomes/5833.html</a> ] and [ <a href="http://www.tigr.org/tdb/CMR/ghi/htmls/SplashPage.html">http://www.tigr.org/tdb/CMR/ghi/htmls/SplashPage.html</a> ]
<i>C. elegans</i> Database	[ <a href="http://www.sanger.ac.uk/Software/Acedb/">http://www.sanger.ac.uk/Software/Acedb/</a> ]
Celera	[ <a href="http://www.celera.com/">http://www.celera.com/</a> ]
dbSNP	[ <a href="http://www.ncbi.nlm.nih.gov/SNP/">http://www.ncbi.nlm.nih.gov/SNP/</a> ]
DDBJ	[ <a href="http://www.ddbj.nig.ac.jp/">http://www.ddbj.nig.ac.jp/</a> ]
ELSI	[ <a href="http://www.nhgri.nih.gov/ELSI/">http://www.nhgri.nih.gov/ELSI/</a> ]
Fly Database (FlyBase)	[ <a href="http://fly.ebi.ac.uk:7081/">http://fly.ebi.ac.uk:7081/</a> ]
GenBank	[ <a href="http://www.ncbi.nlm.nih.gov/Genbank/index.html">http://www.ncbi.nlm.nih.gov/Genbank/index.html</a> ]
GenomeWeb	[ <a href="http://www.hgmp.mrc.ac.uk/GenomeWeb/human-gen-db-genome.html">http://www.hgmp.mrc.ac.uk/GenomeWeb/human-gen-db-genome.html</a> ]
Human Database	[ <a href="http://www.hgmp.mrc.ac.uk/GenomeWeb/human-gen-db-genome.html">http://www.hgmp.mrc.ac.uk/GenomeWeb/human-gen-db-genome.html</a> ]
Incyte	[ <a href="http://www.incyte.com">http://www.incyte.com</a> ]
Jackson Laboratories	[ <a href="http://www.jax.org/">http://www.jax.org/</a> ]
Legume Database (Soybase)	[ <a href="http://genome.cornell.edu/cgi-bin/WebAce/webace?db=soybase">http://genome.cornell.edu/cgi-bin/WebAce/webace?db=soybase</a> ]
Maize Database	[ <a href="http://www.agron.missouri.edu/">http://www.agron.missouri.edu/</a> ]
Mosquito Database	[ <a href="http://klab.agsci.colostate.edu/index.html">http://klab.agsci.colostate.edu/index.html</a> ]
Myriad Genetics	[ <a href="http://www.myriad.com/">http://www.myriad.com/</a> ]
National Agricultural Library	[ <a href="http://www.nal.usda.gov/">http://www.nal.usda.gov/</a> ]
National Human Genome Research Institute	[ <a href="http://www.nhgri.nih.gov/About_NHGRI/">http://www.nhgri.nih.gov/About_NHGRI/</a> ]
NCBI	[ <a href="http://www.ncbi.nlm.nih.gov/">http://www.ncbi.nlm.nih.gov/</a> ]
OMIM	[ <a href="http://www3.ncbi.nlm.nih.gov/omim/">http://www3.ncbi.nlm.nih.gov/omim/</a> ]
Other animal genome databases (ArkDB)	[ <a href="http://www.ri.bbsrc.ac.uk/arkdb/sites.html">http://www.ri.bbsrc.ac.uk/arkdb/sites.html</a> ]
Puffer Fish Database	[ <a href="http://fugu.hgmp.mrc.ac.uk/">http://fugu.hgmp.mrc.ac.uk/</a> ]
The Institute of Genome Research (TIGR):	[ <a href="http://www.tigr.org/">http://www.tigr.org/</a> ]
Tilapia Database	[ <a href="http://www.ri.bbsrc.ac.uk/cgi-bin/arkdb/browsers/browser.sh?species=tilapia">http://www.ri.bbsrc.ac.uk/cgi-bin/arkdb/browsers/browser.sh?species=tilapia</a> ]
Yeast Database	[ <a href="http://genome-www.stanford.edu/">http://genome-www.stanford.edu/</a> ] and [ <a href="http://www.bio.uva.nl/pombe/">http://www.bio.uva.nl/pombe/</a> ]
Zebrafish Database	[ <a href="http://saturn.med.nyu.edu/zfish/pub/">http://saturn.med.nyu.edu/zfish/pub/</a> ]

# Relational databases

Relational systems are best described as being collections of joined tables as in an Excel or Lotus program. A table consists of a fixed number of columns or attributes and a variable number of rows containing cells for single data entry that is relevant to a particular attribute. Several tables of different attributes can be made more useful by linking or joining the cells in each table. In a relational database system the rows of data are presumably limited to the size and capacity of electronic storage with the columns or attributes expandable to include many different characteristics.



# Object-oriented databases

The object-oriented method has proved highly successful in programming applications. An object-oriented database can classify data and use them as specific objects. These classes are arranged in a hierarchical structure that can inherit attributes from higher classes. In a biological system, we may have a class 'experiment' and a specialization of that class called 'in situ hybridization experiment'. This gives object-oriented databases flexibility and extendibility – as genome mapping and sequencing evolves, classes are extended to take into account changing methods and techniques. Object-oriented databases are dynamic and can combine various classes to draw 'virtual' conclusions and generate an independent set of data. One can ask a 'gene object' to give its sequence. Then ask a marker object to draw itself on a genetic map. The modelling of classes controls the results of the application, placing relatively few limitations on the operations.



# Commercial Databases

In addition to the publicly accessible databases there are a number of databases created by private companies for pharmaceutical research and development applications. A few of these companies include:

- Incyte Genomics [<http://www.incyte.com/>],
- Myriad Genetics [<http://www.myriad.com/>], and
- Celera [<http://www.celera.com/>].

# Existing Genome Databases

- Parasites and disease organisms
- Insects
- Plants
- Fish genomes
- Yeast genomes
- Mammalian genomes
- Humans

# Nucleotide Databases

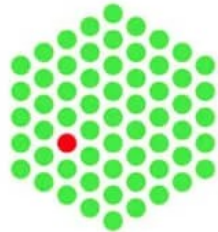
Biological databases store and organize biological data for easy retrieval of information. These centralized resources contain DNA and protein sequences and their associated information.

Nucleotide databases are a type of biological database containing genetic information, which includes DNA and RNA sequences that come from a variety of sources, including whole genomes, transcriptomes, and individual genes.

# Nucleotide Databases

## Nucleotide Databases

EMBL



European Molecular Biology Laboratory



**GenBank**



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



# International Nucleotide Sequence Database

The International Nucleotide Sequence Database Collaboration (INSDC) is a group of three organizations – GenBank, EMBL, and DDBJ – that collect and share nucleotide sequence data.

# GenBank

- GenBank is a sequence database that contains a collection of annotated nucleic acid sequence data.
- It includes various types of genetic material, such as genomic DNA, messenger RNA (mRNA), complementary DNA (cDNA), expressed sequence tags (ESTs), high-throughput raw sequence data, and sequence polymorphisms.

# European Molecular Biology Laboratory (EMBL)

- The European Molecular Biology Laboratory (EMBL) is another nucleotide database, part of the INSDC.
- It is focused on the storage and distribution of nucleotide and protein sequences.
- EMBL also develops tools to help researchers analyze and interpret this data.



# DNA Data Bank of Japan (DDBJ)

- The DNA Data Bank of Japan (DDBJ) is another nucleotide database that exchanges data with GenBank and EMBL as a member of INSDC.
- DDBJ collects and exchanges nucleotide sequence data and manages bioinformatics tools for data submission and retrieval. It also develops tools for biological data analysis and organizes Bioinformatics Training Courses in Japanese.

# Genome Sequence Archive (GSA)

- The Genome Sequence Archive (GSA) is a database that stores raw sequence data and is built based on INSDC data standards and structures.
- GSA was developed to complement the existing INSDC member databases and has now become an important tool for archiving and managing the increasing amount of genomic data.
- GSA accepts raw sequence reads from various sequencing platforms and stores both the sequence reads and metadata submitted by researchers worldwide.
- GSA provides free and unrestricted access to all publicly available data to scientific communities globally.
- GSA uses four primary data objects, namely BioProject, BioSample, Experiment, and Run, to organize the submitted data.

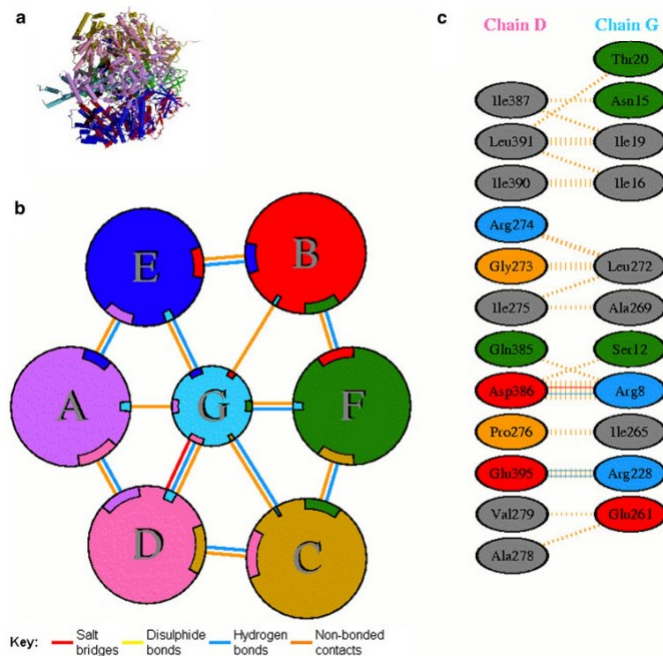
# Single Nucleotide Polymorphism database (dbSNP)

- The Single Nucleotide Polymorphism database (dbSNP) is a public database that contains information about variations in nucleotide sequences.
- It is a part of the National Center for Biotechnology Information (NCBI) and is a public database that accepts entries from both public and private organizations.
- It stores a collection of genetic polymorphisms, including single nucleotide substitutions, deletions or insertions, and microsatellite repeat variations.
- Each entry in dbSNP includes the sequence context of the polymorphism, the occurrence frequency, and the experimental method used to detect the variation.
- dbSNP is open to submissions for variations from any species and genome location.
- The database supports a wide range of research areas, including physical mapping, functional analysis, pharmacogenomics, association studies, and evolutionary studies.

# Nucleic Acid Database (NDB)

- The Nucleic Acid Database (NDB) is a collection of three-dimensional nucleic acid structures and their complexes obtained and curated from the Protein Data Bank (PDB).
- The database acts as a centralized platform for storing and accessing structural information and annotations related to nucleic acids.
- NDB includes annotations specific to the structure and function of nucleic acids. It also provides tools that allow users to search the database, download data and structures, analyze nucleic acids, and learn more about them.
- The database includes RNA and DNA oligonucleotides with two or more bases. It also includes protein-DNA and protein-RNA structures.

# Intermolecular Interactions



Extracts from the protein–protein interaction diagrams in PDBsum for PDB entry 1cow, bovine mitochondrial F1-ATPase. **a** Thumbnail image of the 3D structural model which contains seven protein chains: three of ATPA1\_BOVIN (chains A, B and C), three of ATPB\_BOVIN (chains D, E and F) and a fragment of ATPG\_BOVIN (chain G). **b** Schematic diagram showing the interactions between the chains. The area of each circle is proportional to the surface area of the corresponding protein chain. The extent of the interface region on each chain is represented by a coloured wedge whose colour corresponds to the colour of the other chain and whose size signifies the interface surface area. **c** A schematic diagram showing the residue–residue interactions across one of the interfaces, namely that between chains D and G. Hydrogen bonds and salt bridges are shown as solid lines, while nonbonded contacts are represented by dashed lines (Color figure online)

# Homology model servers

Server	Location	URL
(i) Automatic homology modelling		
3D-JIGSAW	Imperial Cancer Research Fund, UK	<a href="http://www.bmm.icnet.uk/servers/3djigsaw">http://www.bmm.icnet.uk/servers/3djigsaw</a>
CPHmodels	Technical University of Denmark	<a href="http://www.cbs.dtu.dk/services/CPHmodels">http://www.cbs.dtu.dk/services/CPHmodels</a>
ESyPred3D	University of Namur, Belgium	<a href="http://www.fundp.ac.be/urbm/bioinfo/esypred">http://www.fundp.ac.be/urbm/bioinfo/esypred</a>
SWISS-MODEL	Biozentrum Basel, Switzerland	<a href="http://swissmodel.expasy.org">http://swissmodel.expasy.org</a>
(ii) Evaluation of modelling servers		
EVA	Columbia University, USA	<a href="http://cubic.bioc.columbia.edu/eva">http://cubic.bioc.columbia.edu/eva</a>
(iii) Pre-computed homology models		
SWISS-MODEL Repository	Biozentrum Basel, Switzerland	<a href="http://swissmodel.expasy.org/repository">http://swissmodel.expasy.org/repository</a>
ModBase	University of California San Francisco, USA	<a href="http://modbase.compbio.ucsf.edu">http://modbase.compbio.ucsf.edu</a>
PDB archive	wwPDB	<a href="ftp://ftp.wwpdb.org/pub/pdb/data/structures/models">ftp://ftp.wwpdb.org/pub/pdb/data/structures/models</a>



# Review Questions

- Explain the role of genome databases.
- Give some examples of different types of genome databases.





Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 4. Biological Databases

### ☐ Topic 3. Gene databases

#### ☐ Lesson 2. Application of gene databases



ERASMUS+

# APPLICATION OF GENE DATABASES

- Xuhua Xia (2017) Bioinformatics and Drug Discovery. Current Topics in Medicinal Chemistry, 17, 1709-1726
- Benjamin Goudey, Nicholas Geard, Karin Verspoor and Justin Zobel (2022) Propagation, detection and correction of errors using the sequence database network. Briefings in Bioinformatics, 23(6), 1–12
- Van de Sande, B., Lee, J.S., Mutasa-Gottgens, E. et al. (2023) Applications of single-cell RNA sequencing in drug discovery and development. Nat Rev Drug Discov 22, 496–520. <https://doi.org/10.1038/s41573-023-00688-4>



# Contents

- Introduction
- Applications of gene database
- Examples

# Introduction

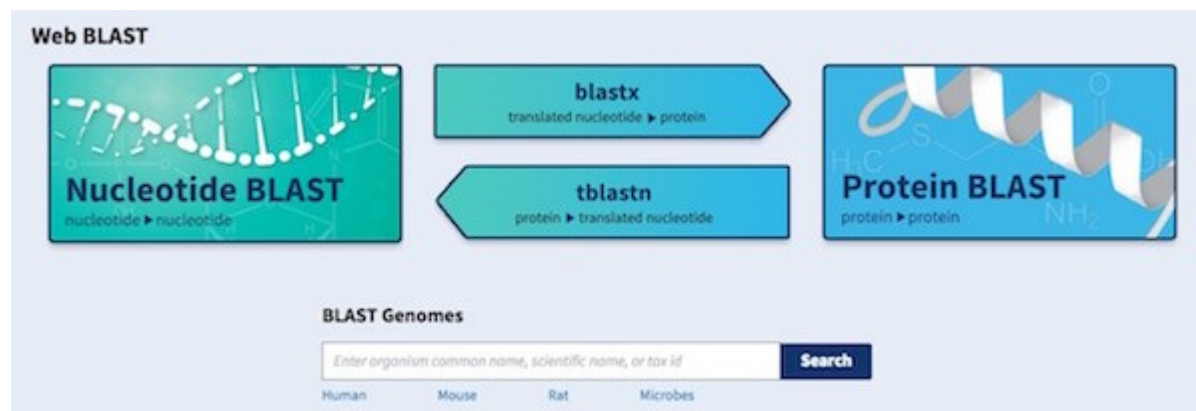
Nowadays researchers explored the uniqueness of DNA sequencing to release the genetic code of numerous diverse organisms to reveal the function of the every organ inside the animal model. From the development of DNA researchers attempted to find the sequencing of complete DNA of many organisms, in some organisms and plants already the whole DNA sequencing genome was established such as human, mouse, rat, bacterial, and plant genomes. From this findings scientist conclude that the most of the biological functions are genetically conserved within and between species, this informs the by gaining the knowledge will helpful to understand the more information about human genome. Sequencing the genomes of diverse organisms brings the greater the intellectual yield DNA sequencing provides significant clue regarding the genes and proteins that are obligatory to generate and sustain related species

# Applications of gene database

- Nucleotide databases are used to identify the gene or the function of a particular nucleotide sequence by comparing an unknown sequence with the known sequences in the database.
- Nucleotide databases can be used to study and examine gene expression by using the sequence information stored in the databases.
- Nucleotide databases are also used to identify potential drug targets and develop new therapies for genetic diseases.
- Nucleotide databases also help in identifying genetic variations that may be linked to diseases, which ultimately helps in the development of diagnostic tools and treatments.
- Nucleotide databases can be used in phylogenetic analysis to analyze the evolutionary relationships between organisms, by comparing and examining their DNA or RNA sequences.

# Basic Local Alignment Search Tool (BLAST)

The Basic Local Alignment Search Tool (BLAST) finds regions of similarity between sequences. The program compares nucleotide or protein sequences and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families.



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



## There are several types of BLAST searches. NCBI's WebBLAST offers four main search types

- **BLASTn** (Nucleotide BLAST): compares one or more nucleotide query sequences to a subject nucleotide sequence or a database of nucleotide sequences. This is useful when trying to determine the evolutionary relationships among different organisms.
- **BLASTx** (translated nucleotide sequence searched against protein sequences): compares a nucleotide query sequence that is translated in six reading frames (resulting in six protein sequences) against a database of protein sequences. Because blastx translates the query sequence in all six reading frames and provides combined significance statistics for hits to different frames, it is particularly useful when the reading frame of the query sequence is unknown or it contains errors that may lead to frame shifts or other coding errors. Thus blastx is often the first analysis performed with a newly determined nucleotide sequence..

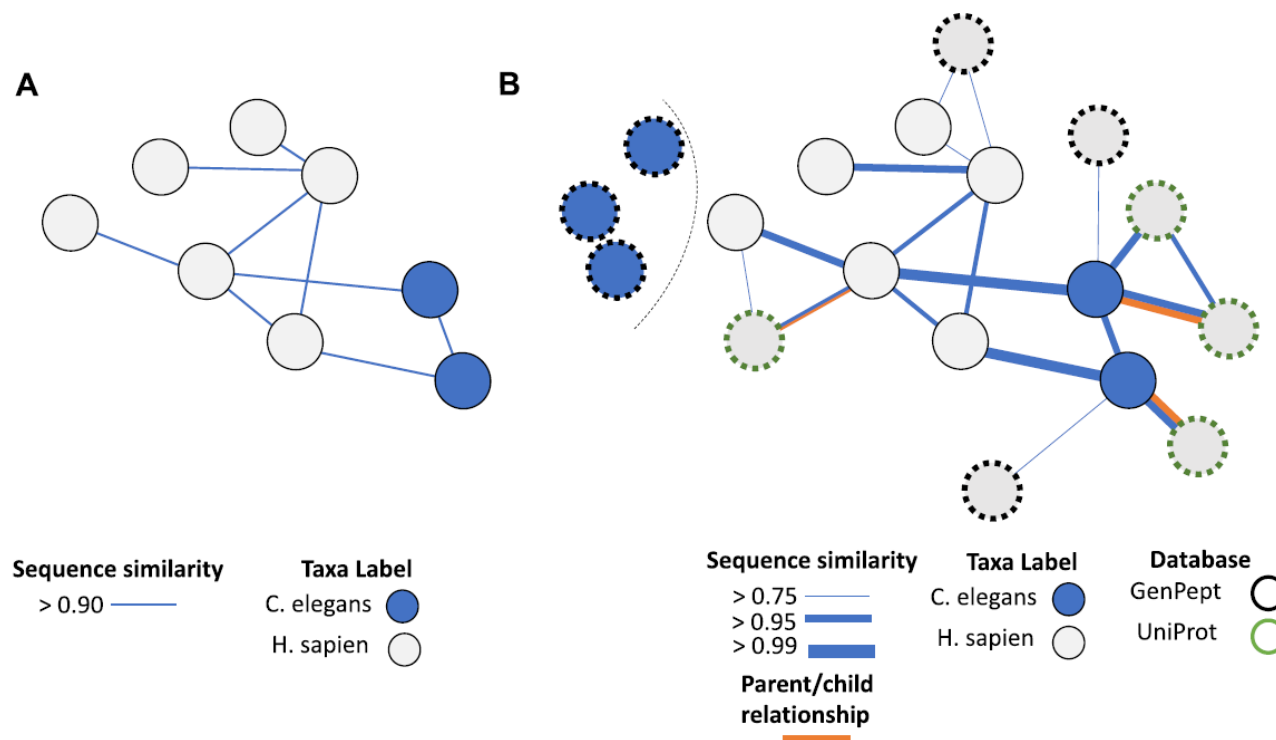
## There are several types of BLAST searches. NCBI's WebBLAST offers four main search types

- **tBLASTn** (protein sequence searched against translated nucleotide sequences): compares a protein query sequence against the six-frame translations of a database of nucleotide sequences. Tblastn is useful for finding homologous protein coding regions in unannotated nucleotide sequences such as expressed sequence tags (ESTs) and draft genome records (HTG), located in the BLAST databases est and htgs, respectively. ESTs are short, single-read cDNA sequences. They comprise the largest pool of sequence data for many organisms and contain portions of transcripts from many uncharacterized genes. Since ESTs have no annotated coding sequences, there are no corresponding protein translations in the BLAST protein databases. Hence a tblastn search is the only way to search for these potential coding regions at the protein level. The HTG sequences, draft sequences from various genome projects or large genomic clones, are another large source of unannotated coding regions.
- **BLASTp** (Protein BLAST): compares one or more protein query sequences to a subject protein sequence or a database of protein sequences. This is useful when trying to identify a protein.

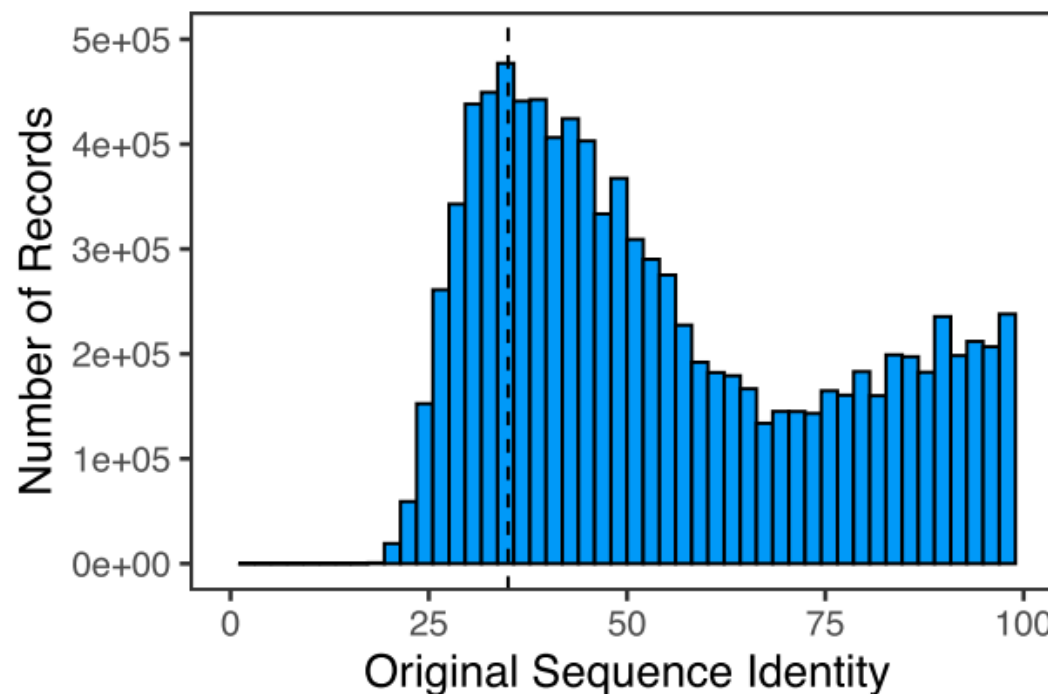
# Sources of errors that affect sequence records, categorized into three broad classes

	Type	Description
Sequence	Assembly errors	Errors in the sequence resulting from poor assembly.
	Contamination	Errors in the sequence resulting from the introduction of foreign material in the sequencing process
Metadata	Sequencing errors	Sequence errors from errors in the sequencing platform.
	Taxonomic misclassification	Incorrect assignment of taxa to a given sequence.
	Functional annotations	Incorrect annotation of the sequence, e.g. incorrectly labelling the function of a protein sequence
Propagation	Annotation boundaries	Inaccurate identification of annotation boundaries, e.g. incorrectly identifying the start or end of a protein in a nucleotide sequence.
	Data entry	Errors in spelling of metadata fields, e.g. incorrect protein names
	Over-prediction	Propagating information from one record to one that is too dissimilar
	Error propagation	Propagating errors from one record to one another
	Staleness	Failing to update a record when source record changes

Example of how a network perspective can help inform outlier detection. (A) A collection of records, with the grey circles indicating records marked as *Caenorhabditis elegans*, while the blue circles are marked as *Homo sapien*. The lines indicate a sequence similarity between records greater than 95%. (B) Expanded network of records



Distribution of sequence identity between bacterial protein records from GenBank that have annotated EC terms and their nearest experimentally validated sequence in UniProt. The black dashed line highlights poor similarity (below 35%), with approximately 2 million records falling below this threshold.



# Nucleotide Databases

Biological databases store and organize biological data for easy retrieval of information. These centralized resources contain DNA and protein sequences and their associated information.

Nucleotide databases are a type of biological database containing genetic information, which includes DNA and RNA sequences that come from a variety of sources, including whole genomes, transcriptomes, and individual genes.



# Network-based anomaly detection

Network-based anomaly detection encompasses a range of techniques to identify records that are different from their local neighbours, varying from clique-analysis to joint-matrix factorization. The approach also offers opportunities to incorporate other sources of knowledge, either making use of the existing manual biocuration approaches in databases such as SwissProt, integrating structured ontologies or knowledge from supporting literature itself, thus integrating information from PubMed articles with the sequence database network. The flexibility of these network approaches and their ability to combine a mixture of different knowledge sources has the potential to enable powerful new tools to detect records that can be flagged as a suspect for manual inspection.



Major types of high-throughput data and their key information relevant to drug discovery. Metabolomic data belong to cheminformatics and are not included.

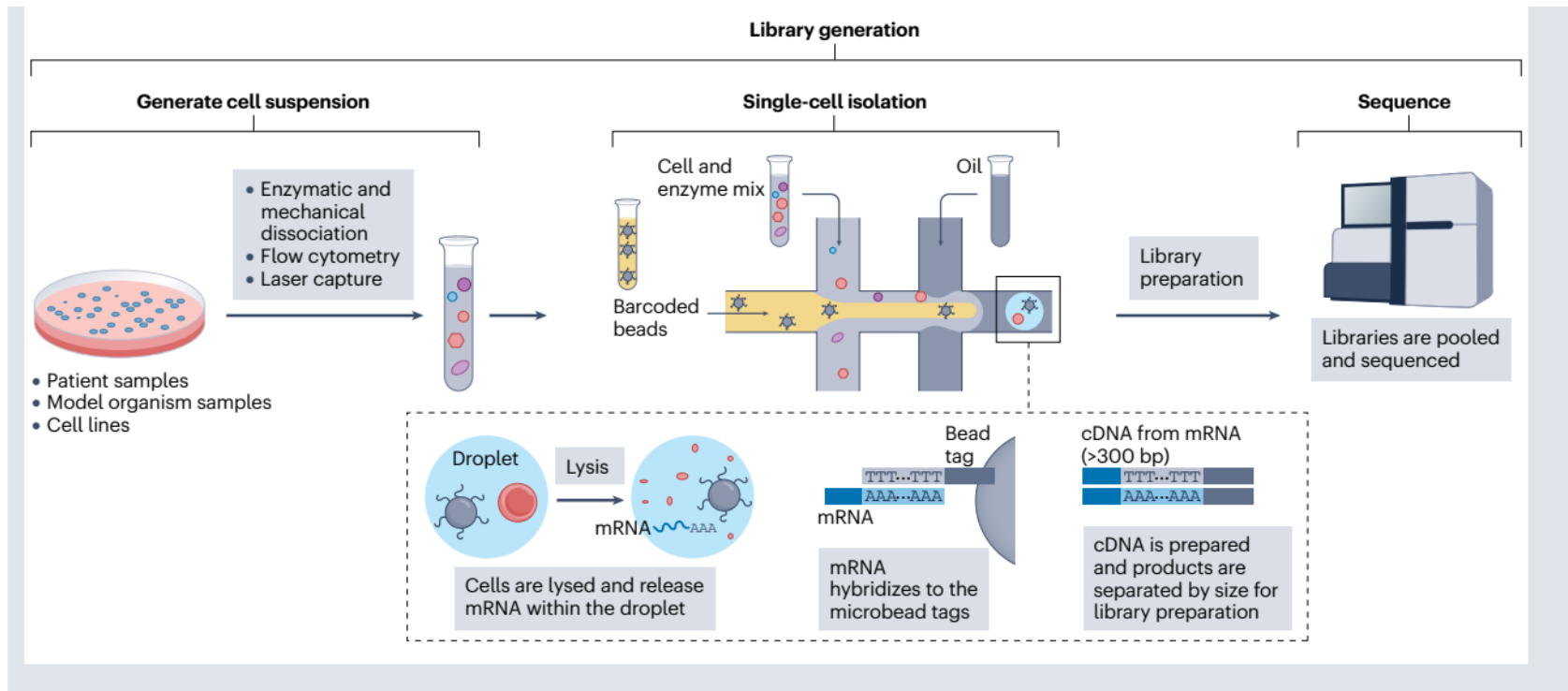


# Disease understanding

As most complex diseases involve multiple cell types, SC (single-cell) resolution can significantly advance disease understanding. ScRNA-seq captures differences in cell-type composition and changes in cellular phenotype that are characteristic of a pathological state. Moreover, the unbiased view of scRNA-seq can detect the presence of rare cell types that drive pathobiology.

SC technologies are providing detailed knowledge of underlying disease mechanisms, enabling the investigation of novel therapeutic approaches.

# Fundamentals of single-cell RNA sequencing



# How single-cell sequencing can inform decisions across the drug discovery and development pipeline

## What is the drug doing at the bedside?

Monitor disease progression and response to therapy with single-cell sequencing to identify mechanisms of action linked to clinically desirable response

## Why do patients present and respond differently?

Identify prognostic factors and markers to stratify patients into actionable subtypes using single-cell methods

Drug response and  
disease progression  
monitoring

Disease subtyping  
and patient  
stratification

Single-cell approaches for drug  
discovery and development

Drug candidate selection

## Which drugs affect disease mechanisms selectively?

Identify drugs matched to disease subtypes with single-cell studies

Target identification  
and credentialling

Preclinical model  
selection

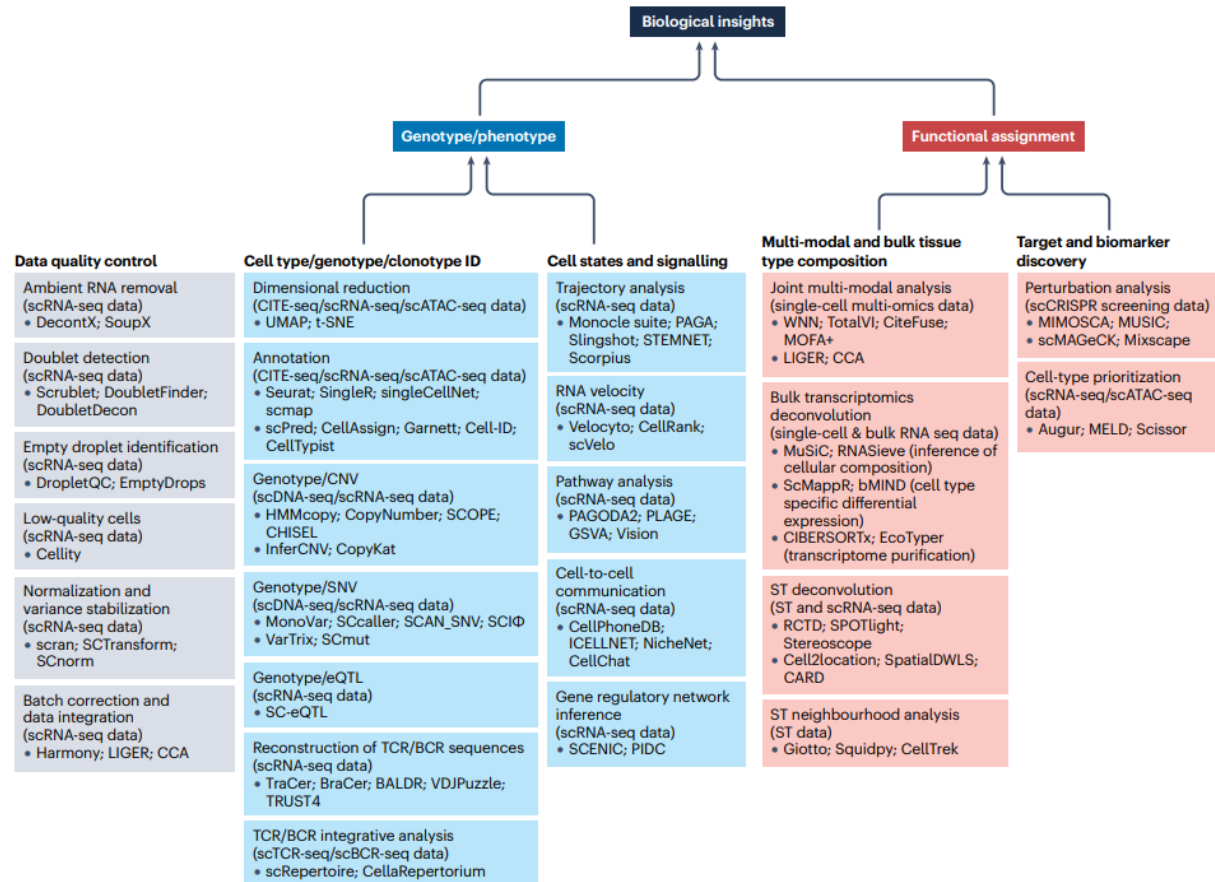
## What drives disease and is suitable for pharmacological intervention?

Identify disease-relevant cell types and targets by using single-cell differential expression analysis and validate them in functional genomics single-cell studies (e.g. CROP-seq, Perturb-seq or CRISPR-seq)

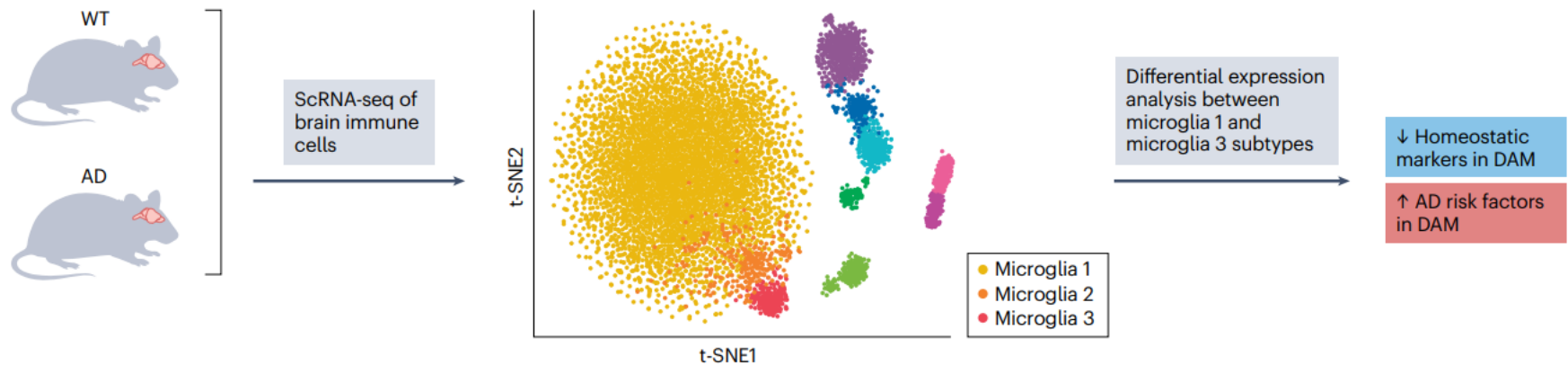
## What can be tested and what are the relevant models?

Identify models at the single-cell level associated to relevant disease subtypes and mechanisms

# Computational methods used in single-cell data analysis for drug discovery and development



# Single-cell RNA sequencing in disease understanding





# Review Questions

- Explain the applications of gene databases.
- Give some examples of different applications of gene databases.





Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 4. Biological Databases

### ❑ Topic 4. KEGG: Kyoto Encyclopedia of Genes and Genomes – high-level functions and utilities of the biological system

#### ❑ Lesson 1. KEGG Database



ERASMUS+

# KEGG Database

- Minoru Kanehisa, Miho Furumichi, Mao Tanabe, Yoko Sato and Kanae Morishima (2017) KEGG: new perspectives on genomes, pathways, diseases and drugs. Nucleic Acids Research, Vol. 45, Database issue D353–D361
- Qiu, YQ. (2013). KEGG Pathway Database. In: Dubitzky, W., Wolkenhauer, O., Cho, KH., Yokota, H. (eds) Encyclopedia of Systems Biology. Springer, New York, NY. [https://doi.org/10.1007/978-1-4419-9863-7\\_472](https://doi.org/10.1007/978-1-4419-9863-7_472)



# Contents

- Introduction
- KEGG Database
- KCF (KEGG Chemical Function) Format
- Reactions
- KEGG Orthology
- Connecting the Chemical and Genetic information

# Introduction

KEGG is a database resource for understanding high-level functions and utilities of the biological system, such as the cell, the organism and the ecosystem, from molecular-level information, especially large-scale molecular datasets generated by genome sequencing and other high-throughput experimental technologies.

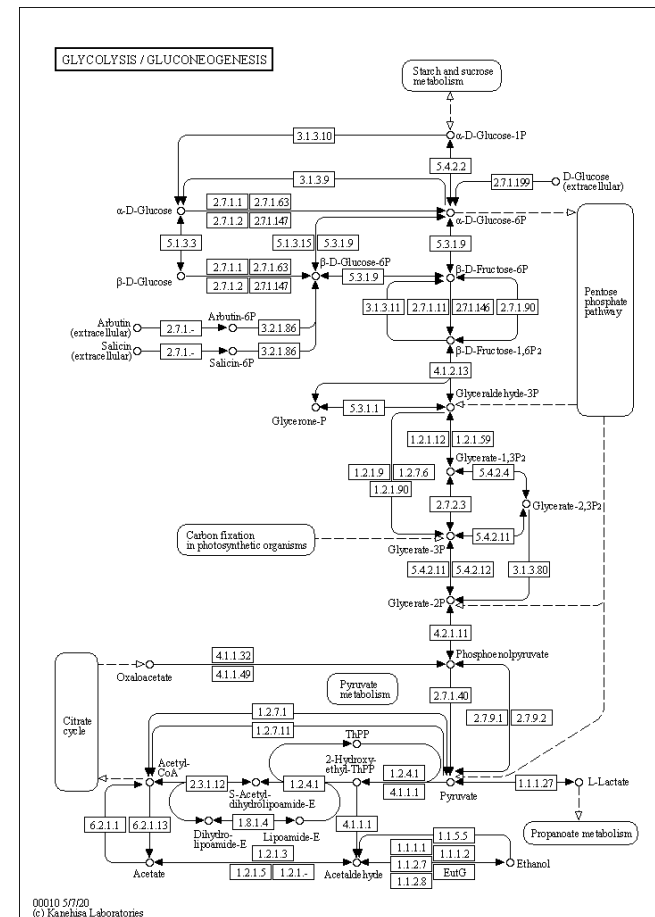
# KEGG Database

- Started in 1995 as the Kyoto Encyclopedia of Genes and
- Genomes (KEGG) database project under the then ongoing
- Human Genome Program in Japan
- Original concept
  - create a reference knowledge base of metabolism and other cellular processes from published literature
  - Why? use for biological interpretation of genome sequence data.
  - How? Create maps of metabolism and annotate with information
  - Examples:
    - ✓ glycolysis → Map 1
    - ✓ citrate acid (TCA) cycle → Map 2
    - ✓ pentose phosphate pathway → Map 3

# Glycolysis / Gluconeogenesis - Reference pathway

[https://www.genome.jp/kegg-bin/show\\_pathway?map=map00010&show\\_description=show](https://www.genome.jp/kegg-bin/show_pathway?map=map00010&show_description=show)

Glycolysis is the process of converting glucose into pyruvate and generating small amounts of ATP (energy) and NADH (reducing power). It is a central pathway that produces important precursor metabolites: six-carbon compounds of glucose-6P and fructose-6P and three-carbon compounds of glyceraldehyde-3P, glycerate-3P, phosphoenolpyruvate, and pyruvate [MD:M00001]. Acetyl-CoA, another important precursor metabolite, is produced by oxidative decarboxylation of pyruvate [MD:M00307]. When the enzyme genes of this pathway are examined in completely sequenced genomes, the reaction steps of three-carbon compounds from glyceraldehyde-3P to pyruvate form a conserved core module [MD:M00002], which is found in almost all organisms and which sometimes contains operon structures in bacterial genomes. Gluconeogenesis is a synthesis pathway of glucose from noncarbohydrate precursors. It is essentially a reversal of glycolysis with minor variations of alternative paths [MD:M00003].





# KEGG now

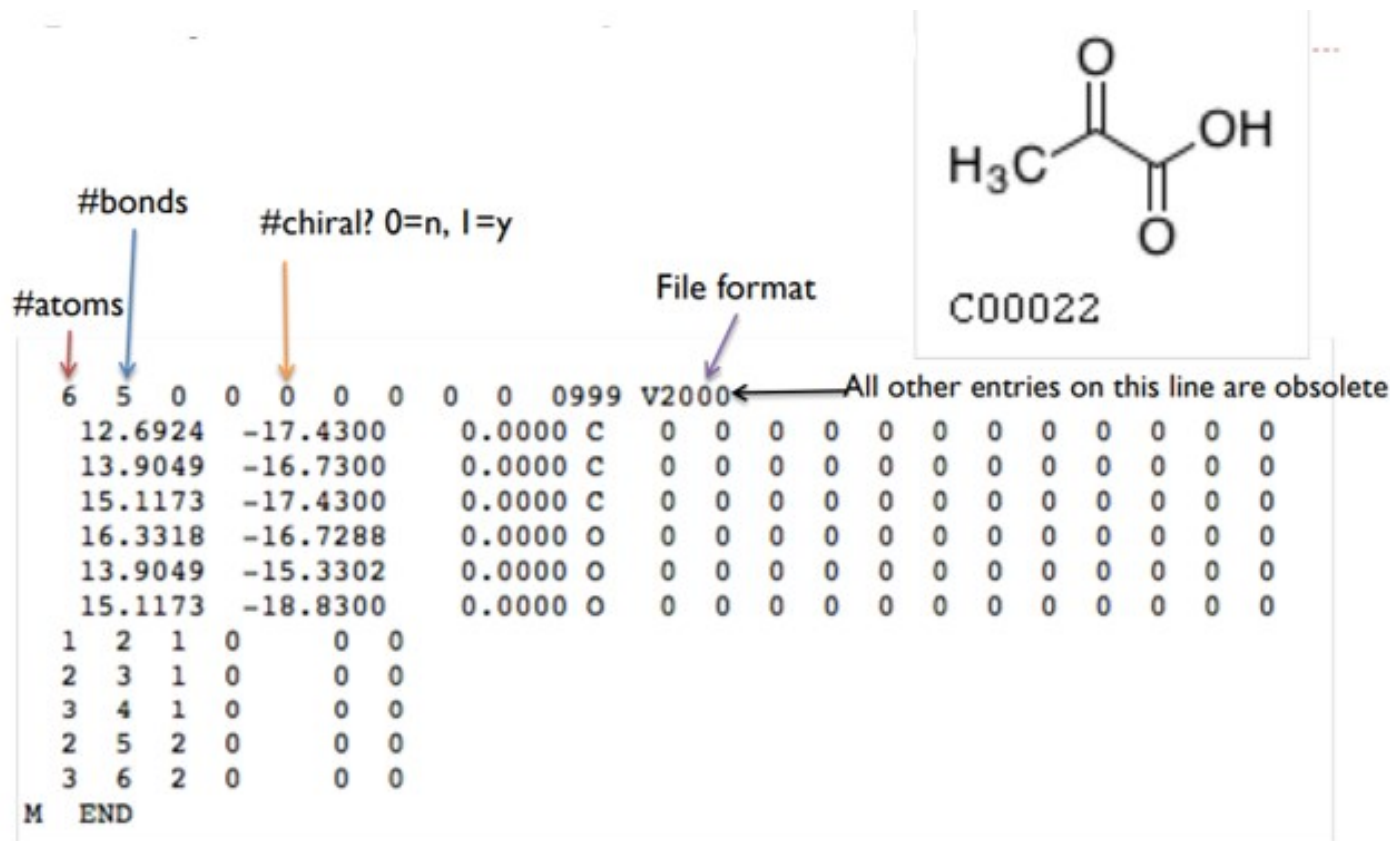
- Integrate Chemistry and Genomics where maps represent different types of networks:
  - chemical networks of how small molecules are converted
    - ✓ Small molecules have molecular weight < 900 daltons, organic, and regulate biological process
  - genomic network of how genome-encoded enzymes are connected to catalyze consecutive reactions
- Maps link:
  - Compounds – represented as circles
  - Reactions – represented as boxes with two identifiers
    - ✓ the reaction identifier (R number)
    - ✓ one or more KO identifiers (K numbers)
  - Yes, we can think of this as a graph, with nodes representing compounds and edges representing reactions



# Mol Files

- List of atoms
- List of bonds
- 2D or 3D spatial coordinates
- Counts of the number of atoms and bonds
- Attributes associated with atoms (charge) or bonds (dashes/wedged bonds,..)
- Attributes associated with an entire structure (e.g. net charge)

# Example Mol file for Pyruvate





# Pros/Cons of Mol files?

## Pros

- Descriptive
- 3D spatial info
- Bond info \*\*

## Cons

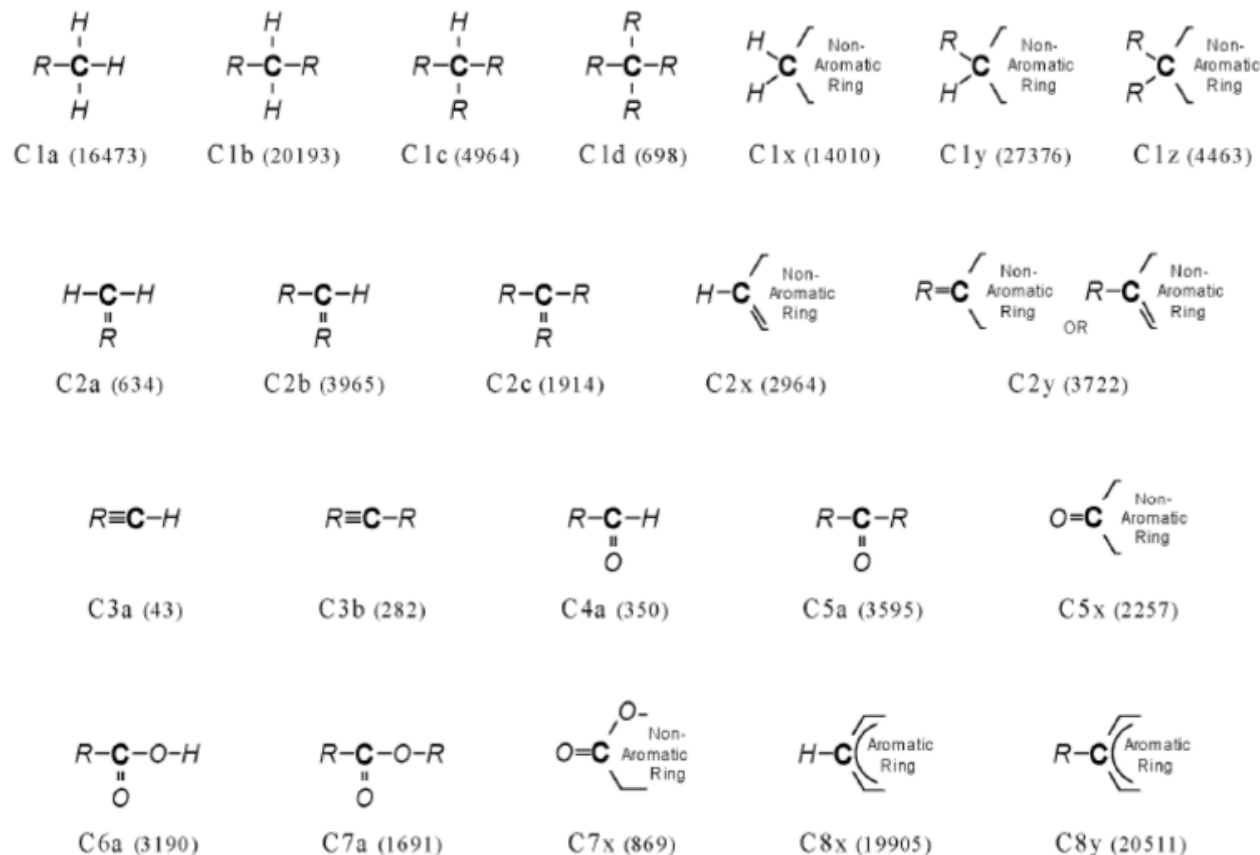
- Non-compact
- Missing H info
- Inefficient for search
- ? For computing bond breaking energy
- Molecular properties like hydrophobia, ..
- Chirality info is not sufficient?
- Amenable to transformation operations
- Extendable

# KCF (KEGG Chemical Function) Format

- Molecules are represented as graphs consisting of non-hydrogen atoms as vertices and bonds as edges
- To encode the molecular environment, vertices (atoms) of KCF are labeled by the 68 KEGG Atom types
  - three-letter labels of the KEGG atoms, such as “C1a” meaning a methyl carbon, represent the hierarchical classification of atom environments.
  - first, the second, and the third letters of the labels are referred to as the “atom species”, the “atom classes”, and the “KEGG atoms”

# KEGG Atom Types for C species

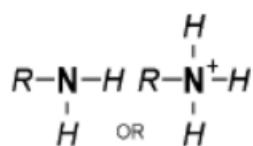
**a**



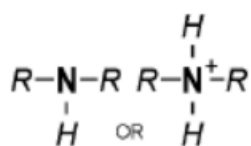


# KEGG Atom Types for N

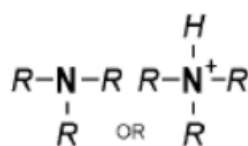
**b**



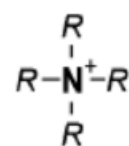
N1a (2440)



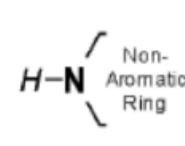
N1b (3003)



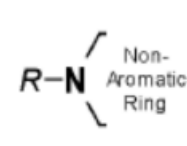
N1c (374)



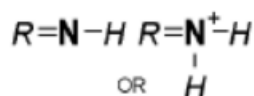
N1d (105)



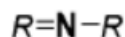
N1x (806)



N1y (1464)



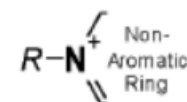
N2a (230)



N2b (163)



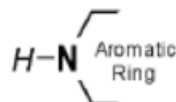
N2x (357)



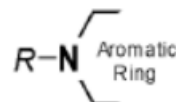
N2y (14)



N3a (119)



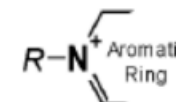
N4x (785)



N4y (840)



N5x (2131)



N5y (59)

# Full Listing of KEGG Atom Types

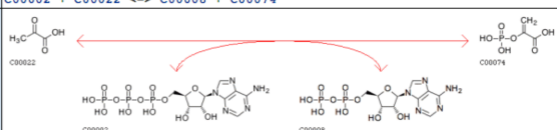
- <http://www.genome.jp/kegg/reaction/KCF.html>

- Detailed drawings:

Hattori, Masahiro, et al. "Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways." *Journal of the American Chemical Society* 125.39 (2003): 11853-11865.

# Reactions

- Reactions –represented in maps as boxes with identifiers
  - EC numbers
  - one or more KO identifiers (K numbers)
  - the reaction identifier (R number)
- Reaction entry:
  - ✓ Name
  - ✓ Definition: Formula in english
  - ✓ Equation
  - ✓ Reaction Class (RC)
  - ✓ Enzyme
  - ✓ Pathway
  - ✓ Orthology
  - ✓ Other DBs

REACTION: R00200	
Entry	R00200
Name	ATP:pyruvate 2-O-phosphotransferase
Definition	ATP + Pyruvate <=> ADP + Phosphoenolpyruvate
Equation	C00002 + C00022 <=> C00008 + C00074
	
Reaction class	RC00002 C00002_C00008 RC00015 C00022_C00074
Enzyme	2.7.1.40
Pathway	rn00010 Glycolysis / Gluconeogenesis rn00230 Purine metabolism rn00620 Pyruvate metabolism rn01100 Metabolic pathways rn01110 Biosynthesis of secondary metabolites rn01120 Microbial metabolism in diverse environments rn01130 Biosynthesis of antibiotics rn01200 Carbon metabolism rn01230 Biosynthesis of amino acids
Module	M00001 Glycolysis (Embden-Meyerhof pathway), glucose => pyruvate M00002 Glycolysis, core module involving three-carbon compounds M00049 Adenine ribonucleotide biosynthesis, IMP => ADP,ATP
Orthology	X00873 pyruvate kinase [EC:2.7.1.40] X12406 pyruvate kinase isozymes R/L [EC:2.7.1.40]
Other DBs	RHEA: 18160

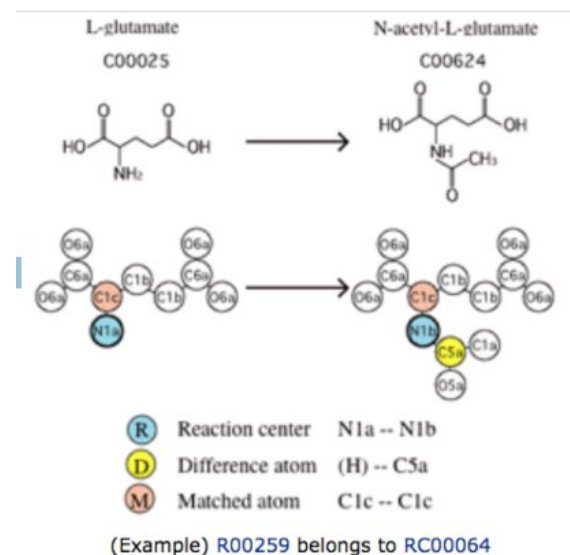
# KEGG Orthology

- The KEGG Orthology (KO) system is a collection of manually defined ortholog groups (KO entries) for all proteins and functional RNAs that appear in the KEGG pathway maps
- What is an ortholog?
  - Genes in different species evolved from a common ancestral gene
  - Typically retain same function
  - Identification of orthologs allow prediction of gene function

# KEGG reaction classes

- Classify reactions based on chemical structure transformation pattern of substrate-products pairs (reaction pairs, or RPAIRS)
- Transformation coded using an RDM (Reaction center, Difference, Match) pattern
- Ex: Acetyl-CoA + L-Glutamate  $\rightleftharpoons$  CoA + N-Acetyl-L-glutamate

A reaction consists of multiple reactant pairs, and the one that appears on the KEGG metabolic pathway map is called the main pair. Currently, the reaction class is defined for each unique RDM pattern or a unique combination of RDM patterns when more than one reaction center is identified for a reactant pair.



# Connecting the Chemical and Genetic information





# Review Questions

- Explain the main features of KEGG.
- Give some examples of using KEGG.





Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 4. Biological Databases

### ❑ Topic 4. KEGG: Kyoto Encyclopedia of Genes and Genomes – high-level functions and utilities of the biological system

#### ❑ Lesson 2. KEGG Software



ERASMUS+

# KEGG Software

- Minoru Kanehisa, Miho Furumichi, Mao Tanabe, Yoko Sato and Kanae Morishima (2017) KEGG: new perspectives on genomes, pathways, diseases and drugs. Nucleic Acids Research, Vol. 45, Database issue D353–D361
- Minoru Kanehisa and others, KEGG for integration and interpretation of large-scale molecular data sets, Nucleic Acids Research, Volume 40, Issue D1, 1 January 2012, Pages D109–D114, <https://doi.org/10.1093/nar/gkr988>



# Contents

- Introduction
- Architecture of KEGG website
- KEGG modules
- Genome annotation in KEGG
- Genome comparison and combination

# Introduction

KEGG is a database resource for understanding high-level functions and utilities of the biological system, such as the cell, the organism and the ecosystem, from molecular-level information, especially large-scale molecular datasets generated by genome sequencing and other high-throughput experimental technologies.

# Architecture of KEGG website

Layer	Content
Top pages	KEGG home ( <a href="http://www.kegg.jp">www.kegg.jp</a> ) Release notes, statistics, database/software documents, KEGG API, KGML
DB entry points	KEGG2 page for table of contents ( <a href="http://www.kegg.jp/kegg/kegg2.html">www.kegg.jp/kegg/kegg2.html</a> ) Data-oriented entry points (corresponding to Table 1) Subject-oriented entry points (shown in Table 5) Organism-specific entry points (for individual genome, multiple genomes, pangenome, organism group)
DB contents	Database entries (as those shown in Table 2)
Software tools <sup>a</sup>	KEGG Mapper tools ( <a href="http://www.kegg.jp/kegg/mapper.html">www.kegg.jp/kegg/mapper.html</a> ) BlastKOALA automatic annotation server ( <a href="http://www.kegg.jp/blastkoala/">www.kegg.jp/blastkoala/</a> ) GhostKOALA automatic annotation server ( <a href="http://www.kegg.jp/ghostkoala/">www.kegg.jp/ghostkoala/</a> )



#### KEGG Home

Release notes  
Current statistics

#### KEGG Database

KEGG overview  
Searching KEGG  
KEGG mapping  
Color codes

#### KEGG Objects

Pathway maps  
Brite hierarchies  
KEGG DB links

#### KEGG Software

KEGG API  
KGML

#### KEGG FTP

Subscription  
Background info

#### GenomeNet

#### DBGET/LinkDB

Feedback  
Copyright request

#### Kanehisa Labs

## KEGG Software

### Web Servers

The following web tools are developed and maintained by Kanehisa Laboratories.

- BlastKOALA** KOALA family tools for automatic annotation of genome and metagenome sequences with subsequent KEGG Mapper analysis. [\[reference\]](#)
- GhostKOALA**
- KEGG Mapper** KEGG mapping against PATHWAY/BRITE/MODULE databases for biological interpretation of genomic, transcriptomic, metabolomic, and other large-scale data sets. [\[reference\]](#)

The following web servers are developed and maintained by Kyoto University Bioinformatics Center as part of its GenomeNet service.

- KofamKOALA** Another KOALA family tool for automatic KO assignment and KEGG mapping.
- KAAS** The original KEGG automatic annotation server. [\[reference\]](#)
- KEGG OC** KEGG OC viewer for browsing and analyzing ortholog clusters (OCs) computationally generated from the KEGG SSDB database. [\[reference\]](#)
- SIMCOMP** Chemical structure similarity search against KEGG COMPOUND, KEGG DRUG, and other databases. SIMCOMP is based on 2D graph representation, while SUBCOMP is based on bit-string representation of chemical structures. [\[references\]](#)
- SUBCOMP**
- KCaM** Glycan structure similarity search against KEGG GLYCAN using tree structure comparison methods. [\[reference\]](#)
- E-zyyme** Computational assignment of EC number sub-subclasses from chemical structure transformation patterns of substrates and products. [\[references\]](#)
- PathPred** Prediction of microbial biodegradation pathways and plant second metabolite biosynthesis pathways using reaction patterns in KEGG RCLASS. [\[references\]](#)
- GENIES** Gene network prediction from heterogeneous data sets using kernel methods and partially known network information. [\[references\]](#)
- DINIES** Drug-target interaction network prediction from various types of biological data including chemical structures, drug side effects, amino acid sequences and protein domains. [\[references\]](#)

The following service is also available at GenomeNet.

- BLAST** Sequence similarity search against KEGG GENES, KEGG GENOME, and other databases.
- FASTA**

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



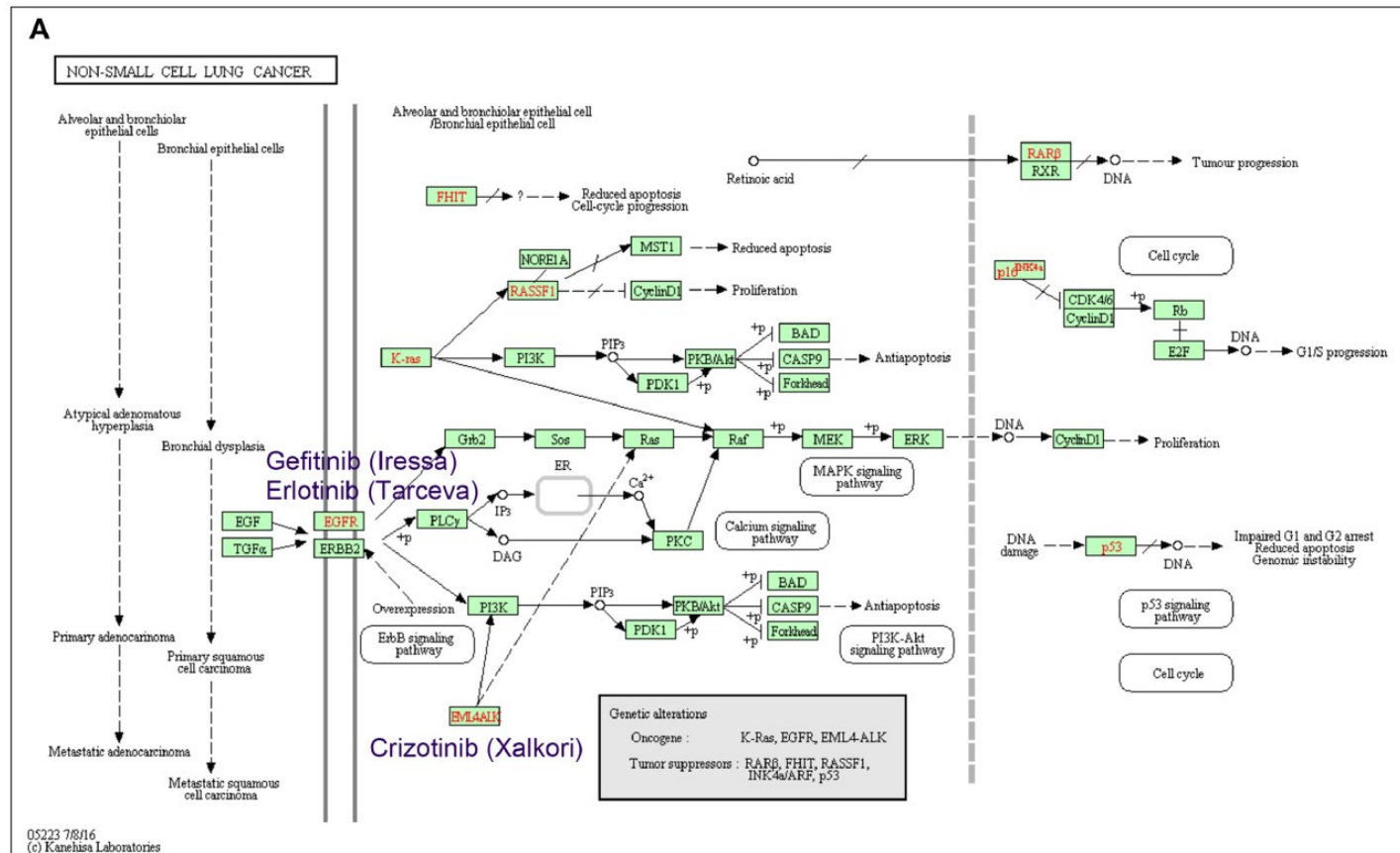
# BlastKOALA tools

BlastKOALA is the web server for automatic annotation (KO assignment) of query amino acid sequences followed by KEGG Mapper analysis for inferring higher-level functions. The server makes full use of the improvements made for the GENES and KO databases, including the addition of the GENES addendum category, the precise taxonomic classification of GENES data and the improvement of KO to sequence links. The taxonomic classification was used to define ‘non-redundant’ GENES datasets for BlastKOALA.

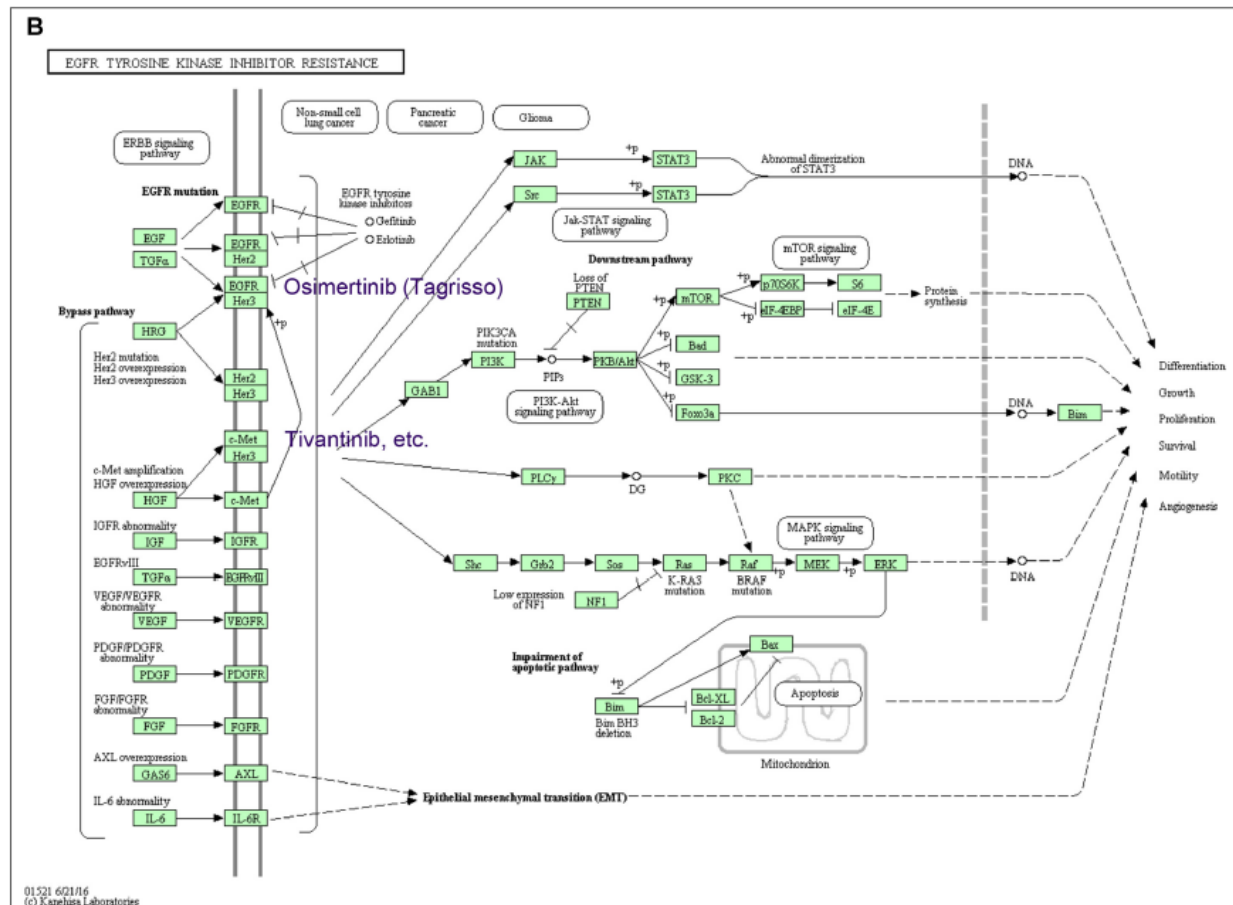
# KEGG mapper tools

KEGG Mapper is a collection of KEGG mapping tools for linking molecular objects (genes, proteins, metabolites and glycans) to higher-level objects (pathways, modules, hierarchies, taxonomy and diseases). Table 7 shows the current list of KEGG mapping tools including the new Search Disease tool. Two pathway mapping tools, Search Pathway and Search&Color Pathway, were made available from the beginning of the KEGG project, and they are still the most widely used. The mapping of genes and proteins can be made either in reference mode using KOs or in organismspecific mode using gene identifiers, such as human gene identifiers used to search against human pathway maps. The three tools, Reconstruct Pathway, Reconstruct Brite and ReconstructModule, acceptKOs only. They are linked from the BlastKOALA and GhostKOALA servers for interpretation of the KO assignment results.

# KEGG pathway maps for non-small cell lung cancer (hsa05223)



# KEGG pathway maps for (B) EGFR tyrosine kinase inhibitor resistance (hsa01521).



# KEGG Mapper tools

Tool	Query dataset	Database
Search Pathway	KOs, gene identifiers, C numbers, etc.	PATHWAY
Search&Color Pathway	KOs, gene identifiers, C numbers, etc.	PATHWAY
Color Pathway	KOs, gene identifiers	single KEGG pathway map
Color Pathway WebGL	KOs, gene identifiers	single KEGG pathway map
Search Brite	KOs, gene identifiers, C numbers, etc.	BRITE
Search&Color Brite	KOs, gene identifiers, C numbers, etc.	BRITE
Join Brite	KOs, D numbers, etc.	single BRITE hierarchy
Join Brite Table	KOs, D numbers, etc.	single BRITE table
Search Module	KOs, gene identifiers, C numbers, etc.	MODULE
Search&Color Module	KOs, gene identifiers, C numbers, etc.	MODULE
Search Disease	KOs, human gene identifiers	DISEASE
Reconstruct Pathway	KOs	PATHWAY
Reconstruct Brite	KOs	BRITE
Reconstruct Module	KOs	MODULE
Map Taxonomy	Organism codes, NCBI taxonomy IDs	Taxonomy file

# KEGG modules

KEGG MODULE was originally introduced to define tighter functional units than KEGG PATHWAY so that the pathway information in KEGG would be represented in three resolutions: global maps (for metabolism), regular maps and modules. Subsequently, the scope of KEGG MODULE has been extended and there are currently four types of KEGG modules:

- (i) pathway modules,
- (ii) structural complexes,
- (iii) functional sets and
- (iv) signature modules.

The first three types of modules usually correspond to parts of KEGG pathway maps and BRITE hierarchies. The signature module is a set of genes in the genome, or perhaps in the transcriptome as well, that can be used as a marker for the phenotype, such as pathogenicity and metabolic capacity. Each KEGG module is defined by the combination of K numbers and associated with an automatically generated module map according to the predefined notations: space delimited items for pathway elements, comma separated items in parentheses for alternatives, plus sign to define a complex and minus sign for an optional item.



# Genome annotation in KEGG

The genome annotation in KEGG is essentially cross-species annotation, finding orthologous genes in all available genomes for given K numbers, and is currently performed as follows.

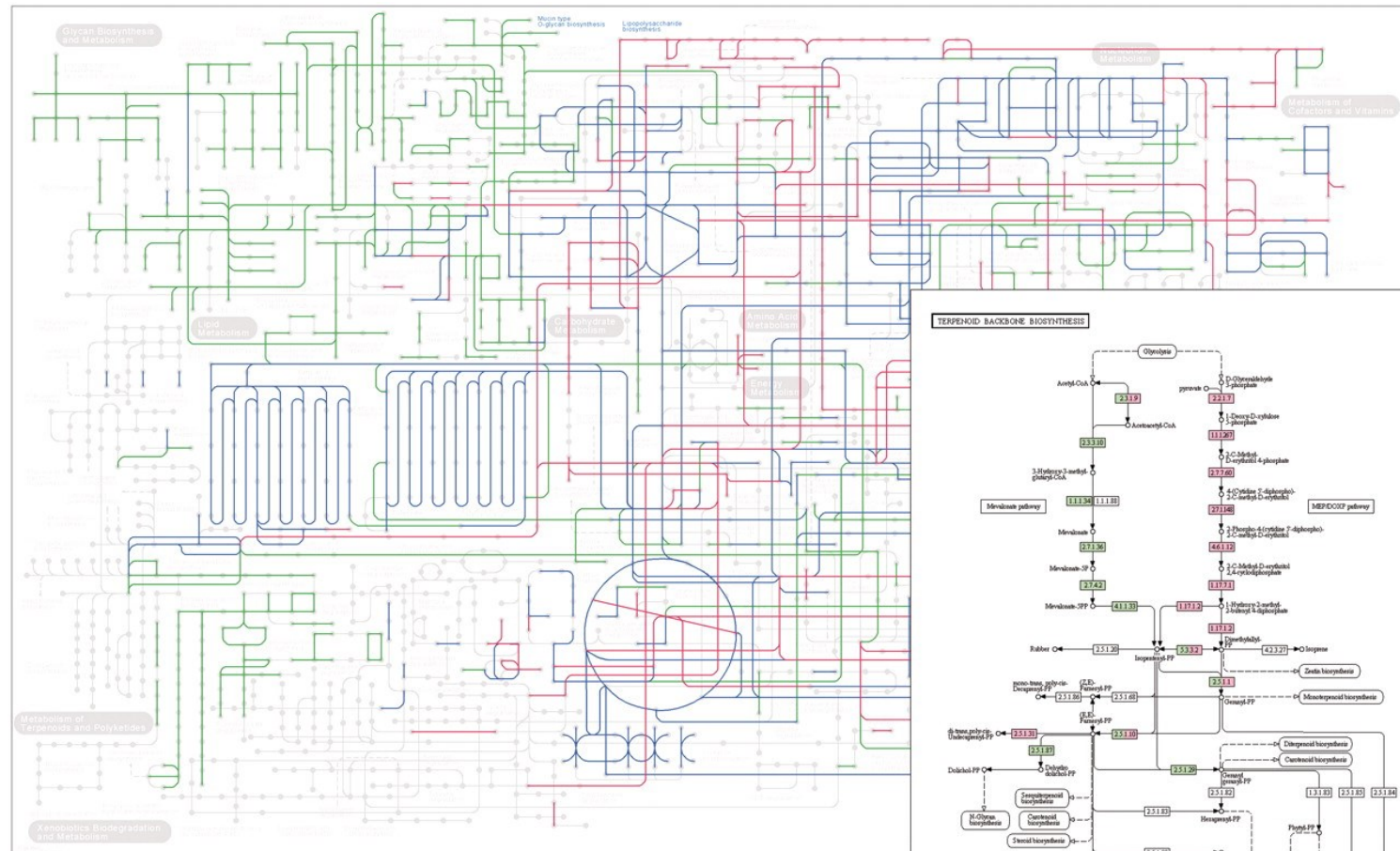
- (i) Experimental evidence on known functions of genes and proteins is organized in the KO database, which is created together with the KEGG PATHWAY, BRITE and MODULE databases.
- (ii) Gene catalogs of complete genomes are generated from RefSeq and other public resources.
- (iii) All pairs of genomes (gene catalogs) are compared by the SSEARCH program, and the GFIT tables are generated detailing the information for each gene in a genome about best-hit genes in all other genomes.
- (iv) GFIT tables are continuously updated, and the automatic version of the KOALA tool presents to human annotators a summary of discrepancies between its K number assignment and the current annotation.
- (v) Discrepancies are examined by annotators with the manual version of KOALA and other tools such as for protein domains, ortholog tables and gene clusters. We plan to incorporate KEGG modules into the KOALA annotation procedure.



# Genome comparison and combination

For the organisms (GENES, DGENES and EGENES) and environmental samples (MGENES) available in KEGG, it is now possible to map multiple data sets against KEGG pathway maps and BRITE functional hierarchies. The user interface may be found in the KEGG GENOME page, and the result is displayed using multiple color coding. This feature can be used, for example, to compare metabolic capabilities of different organisms, to examine complementarity of host–symbiont, host–pathogen and host–microbiome relationships, and to examine collective features of pangenomes. These tasks may be applied to the user's own genomes by using KEGG Mapper with preprocessing of K number assignment and color specification.

# Comparison of metabolic pathways reconstructed from the complete genomes of Homo sapiens (hsa) and E. coli (eco)



ERASMUS+

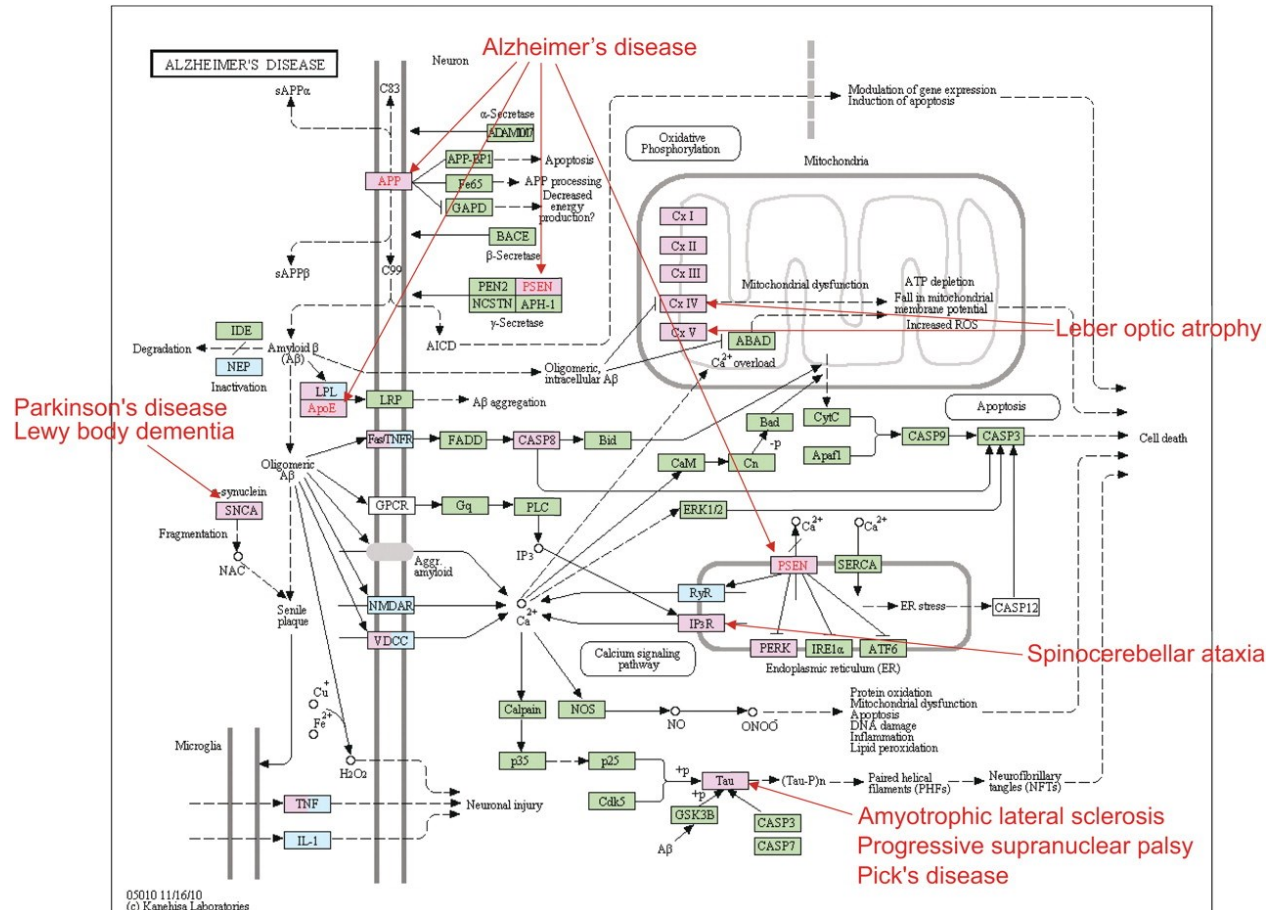
Key Action KA2 - Cooperation for innovation and the exchange of good practices

Action Type KA226 - Partnerships for Digital Education Readiness

# Knowledge base extension

The mapping of disease and drug data from these databases is now incorporated in the daily KEGG update procedure. First, all known disease genes accumulated in KEGG DISEASE and all known drug targets accumulated in KEGG DRUG are integrated in the KEGG PATHWAY and BRITE databases. This is accomplished by preparing binary relations, human gene identifier to H number and human gene identifier to D number, as query data sets and applying the Search&Color Pathway and Search&Color Brite tools. The generated pathway maps and BRITE hierarchy files are identified by the 'hsadd' prefix and the '\_dd' modifier, respectively. The mapping result is displayed by coloring: pink when the gene is associated with a disease and light blue when the gene product is a drug target.

Disease/drug mapping is the process to map all known disease genes (pink) and all known drug targets (light blue) against all KEGG pathway maps



# Naming convention of KEGG molecular networks

Molecular network	Manually created		Computationally generated		
	Reference		Organism-specific	Disease/drug	Other
Metabolic pathway	map00010	ko00010	hsa00010	hsadd00010	
		ec00010			
		rn00010			
Non-metabolic pathway	map02010	ko02010	hsa02010	hsadd02010	
Gene/protein brite	ko02000		hsa02000	hsa02000_dd	
Non-gene/protein brite	br08303				br08303_target
					br08303_enzyme
Module	M00001		hsa_M00001		



# Accessing KEGG

KEGG is made available at both the GenomeNet website (<http://www.genome.jp/kegg/>) and the KEGG website (<http://www.kegg.jp/>). The GenomeNet site has been the primary site, but this will change at the end of 2011. The KEGG site will then be the primary site handling all database update procedures, and the GenomeNet site will become a mirror site.



# Review Questions

- Explain the architecture of KEGG website.
- Give some examples of using KEGG.





Tackling COVID-19 – Challenges Aiming at Bettering the Higher  
Education Quality (TACOHEQ)  
Project Number: 2020-1-BG01-KA226-HE-095131

Co-funded by the  
Erasmus+ Programme  
of the European Union



# Thank you for your attention!

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 5. Practice

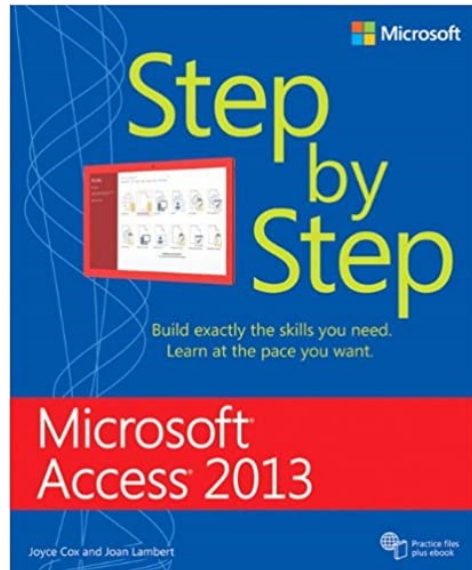
### ❑ Topic 1. Introduction to the computer program MS Access. Use of tables, subforms, filters and reports.

#### ❑ Practical lesson 1. Basic concepts in databases. Introduction to the computer program MS Access.

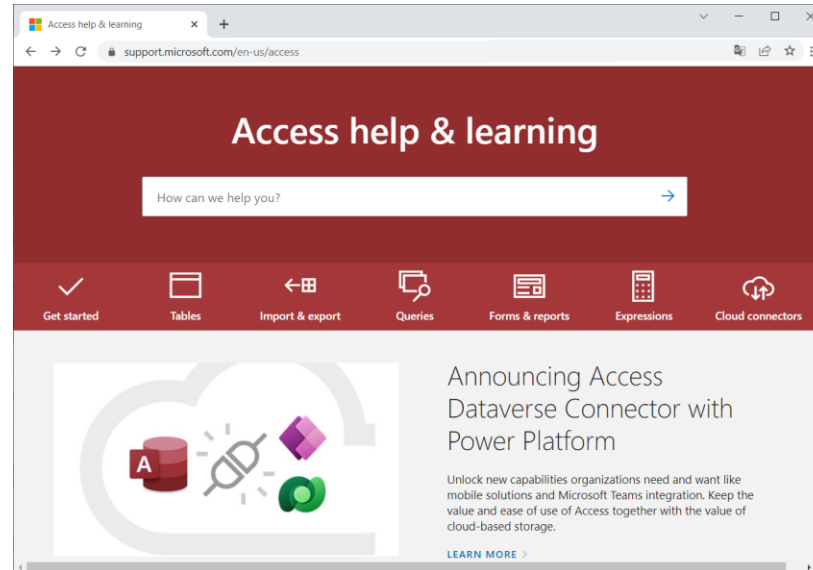


ERASMUS+

# Practical lesson. Basic concepts in databases. Introduction to the computer program MS Access.



*Joyce Cox, Joan Lambert.  
Microsoft Access 2013 Step By  
Step, Microsoft Press., 2013.*



*<https://support.microsoft.com/en-us/access>*

## What is a database?

- A **database** is a tool for collecting and organizing information. Databases can store information about people, products, orders, or anything else.
- Many databases start as a list in a word-processing program or spreadsheet. As the list grows bigger, redundancies and inconsistencies begin to appear in the data.
- The data becomes hard to understand in list form, and there are limited ways of searching or pulling subsets of data out for review. Once these problems start to appear, it's a good idea to transfer the data to a database created by a database management system (DBMS), such as **MS Access**.

## What is a database?

- **Tables** are one of the types of database objects you work with in **Access**.
- Other types include forms, queries, reports, macros, and modules.
- Of these object types, only tables are used to store information.
- The others are used to enter, manage, manipulate, analyze, retrieve, or display the information stored in tables—in other words, to make the information as accessible and therefore as useful as possible.

## What is a database?

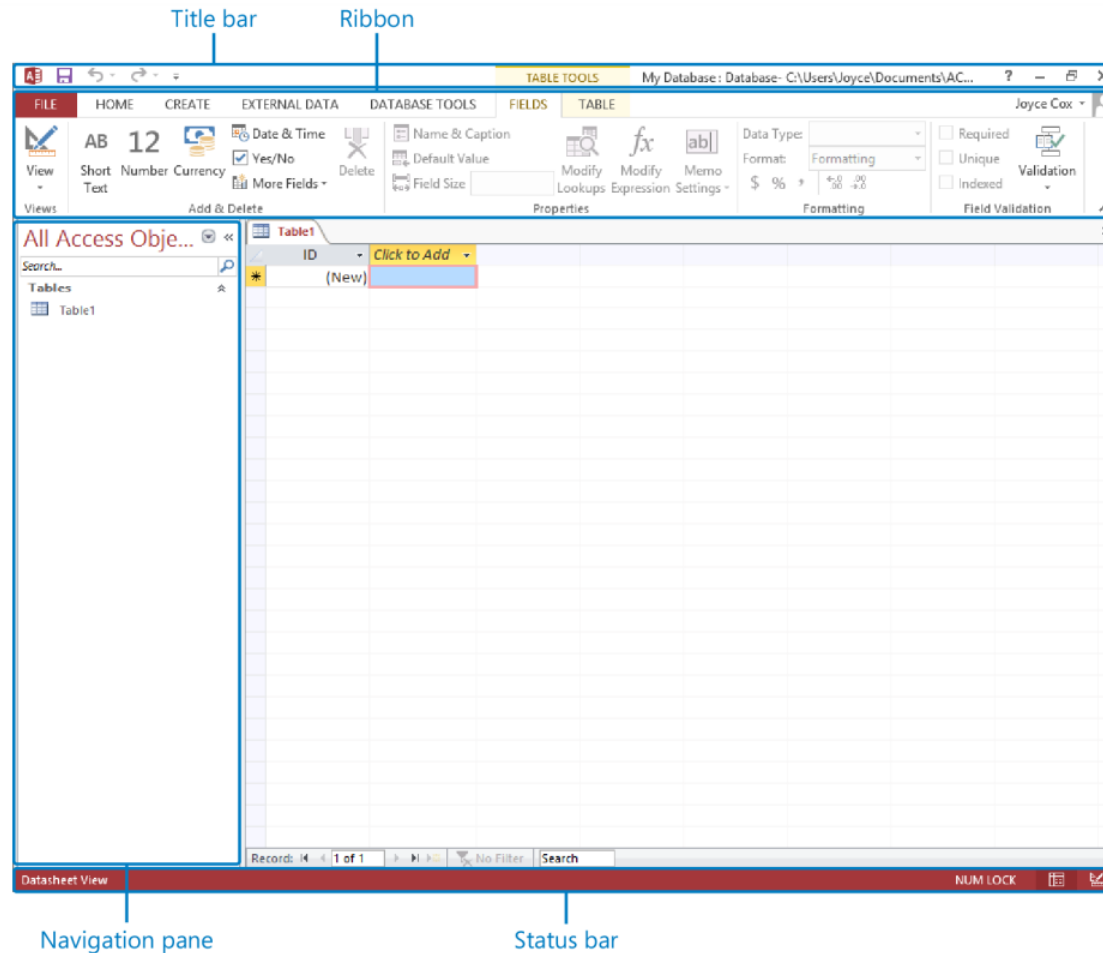
- In its most basic form, a database is the electronic equivalent of an organized list of information. Typically, this information has a common subject or purpose, such as the list of employees shown in the following table.

ID	First name	Last name	Title	Hire date
1	Karen	Berg	Owner	May 1, 2008
2	Kim	Akers	Head Buyer	June 1, 2008
3	Tom	O'Neill	Assistant	November 2, 2008
4	Naoki	Sato	Sales Manager	August 14, 2009
5	Molly	Dempsey	Gardener	October 17, 2009
6	Nancy	Anderson	Sales Rep	May 1, 2010
7	Michael	Entin	Sales Rep	April 1, 2011
8	Kari	Furse	Buyer	May 3, 2011
9	Chase	Carpenter	Gardener	November 15, 2012

The real power of a database isn't in its ability to store information; it is in your ability to quickly retrieve exactly the information you want from the database.

- This list is arranged in a table of columns and rows. Each column represents a field—a specific type of information about an employee: last name, first name, hire date, and so on. Each row represents a record—all the information about a specific employee.

# Introduction to MS Access



*A new blank table displayed in the Access 2013 program window.*

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness



# Introduction to MS Access

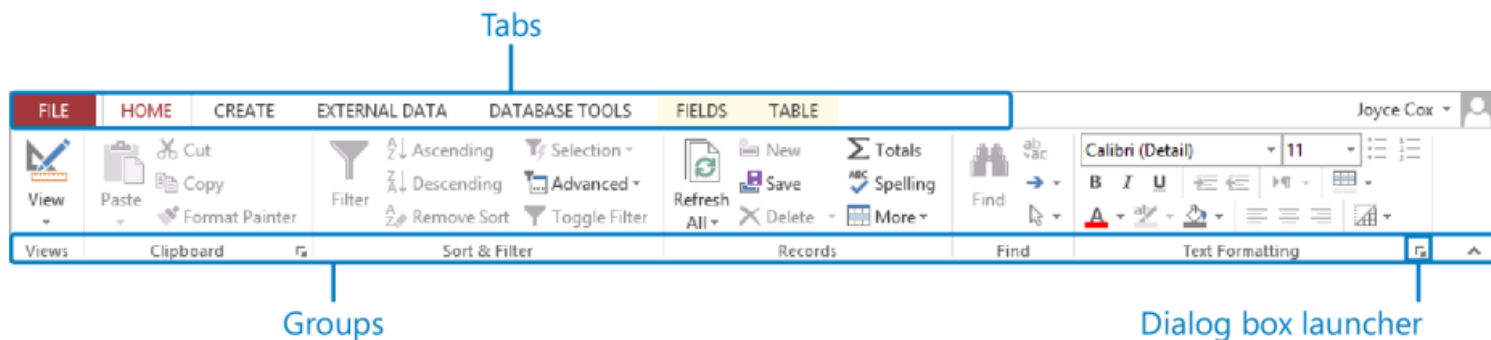
- The program window contains the following elements:
  - **Title bar.** This bar across the top of the program window displays the name of the active database and by default display the path to the folder where it is stored. It also provides tools for managing the program and the program window.



You can use the tools on the title bar to move and size the window, undo or redo changes, save the database, and get help with the program. At the left end of the title bar is the program icon, which you click to display commands to restore, move, size, minimize, maximize, and close the program window. To the right of the Access icon is the Quick Access Toolbar.

# Introduction to MS Access

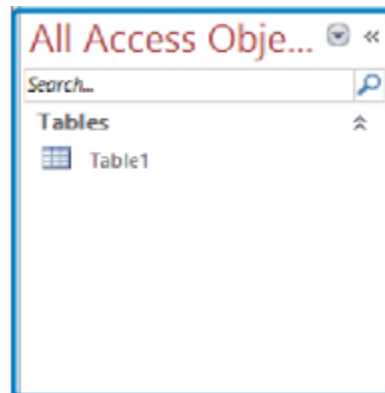
- The program window contains the following elements:
  - Ribbon.** Below the title bar, all the commands for working with an Access database are represented as buttons in this central location so that you can work efficiently with the program.



Each tab of the ribbon contains a specific category of commands.

## Introduction to MS Access

- The program window contains the following elements:
  - **Navigation pane.** On the left side of the program window, the Navigation pane displays lists of database objects. By default, it displays all the objects in the database by type of object, but you can filter the list by clicking the pane's title bar and then clicking the category or group of objects you want to display. expand the groups in the list by clicking the chevrons in the section bars..



KEYBOARD SHORTCUT Press F11 to display or hide the Navigation pane.

## Introduction to MS Access

- The program window contains the following elements:
  - **Status bar.** Across the bottom of the program window, this bar displays information about the current database and provides access to certain program functions. At the right end of the bar is the View Shortcuts toolbar, which provides convenient buttons for switching the view of the active database object.



*This status bar tells you the current view and the status of the keyboard.*

**The goal of all these user interface features is to make working in a database as intuitive as possible. Commands for tasks you perform often are readily available, and even those you might use infrequently are easy to find.**

## Introduction to MS Access

- Using MS Access, you can:
  - Add new data to a database, such as a new item in an inventory
  - Edit existing data in the database, such as changing the current location of an item
  - Delete information, perhaps if an item is sold or discarded
  - Organize and view the data in different ways
  - Share the data with others via reports, e-mail messages, an intranet , or the Internet

## Introduction to MS Access

- The following sections are short descriptions of the parts of a typical Access database.
  - Tables
  - Forms
  - Reports
  - Queries
  - Macros
  - Modules

# Introduction to MS Access

## • Tables



- A **database** table is similar in appearance to a spreadsheet, in that data is stored in **rows** and **columns**. As a result, it is usually quite easy to import a spreadsheet into a database table.
- To get the most flexibility out of a database, the data needs to be organized into tables so that redundancies don't occur.
- For example, if you're storing information about employees, each employee should only need to be entered once in a table that is set up just to hold employee data.
- Data about products will be stored in its own table, and data about branch offices will be stored in another table. This process is called normalization.



# Introduction to MS Access

## • Tables

- Each row in a table is referred to as a **record**. Records are where the individual pieces of information are stored. Each record consists of one or more fields.
- **Fields** correspond to the columns in the table. For example, you might have a table named "Employees" where each record (row) contains information about a different employee, and each field (column) contains a different type of information, such as first name, last name, address, and so on.
- Fields must be designated as a certain **data type**, whether it's text, date or time, number, or some other type.

# Introduction to MS Access

- **Forms**



- **Forms** allow you to create a user interface in which you can enter and edit your data.
- Forms often contain command buttons and other **controls** that perform various tasks.
- You can create a database without using forms by simply editing your data in the table **datasheets**.
- However, most database users prefer to use forms for viewing, entering, and editing data in the tables.
- You can program command buttons to determine which data appears on the form, open other forms or reports, or perform a variety of other tasks.

# Introduction to MS Access

- **Forms**



- For example, you might have a form named "Customer Form" in which you work with customer data.
- The customer form might have a button which opens an order form where you can enter a new order for that customer.
- Forms also allow you to control how other users interact with the data in the database.
- For example, you can create a form that shows only certain fields and allows only certain operations to be performed. This helps protect data and to ensure that the data is entered properly.

## Introduction to MS Access

- **Reports**



- **Reports** are what you use to format, summarize and present data.
- A report usually answers a specific question, such as "How much money did we receive from each customer this year?" or "What cities are our customers located in?"
- Each report can be formatted to present the information in the most readable way possible.
- A report can be run at any time, and will always reflect the current data in the database.
- Reports are generally formatted to be printed out, but they can also be viewed on the screen, exported to another program, or sent as an attachment to an e-mail message.

# Introduction to MS Access

## • Queries



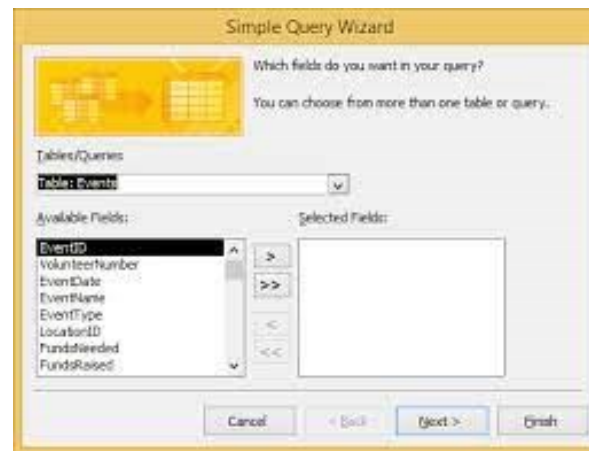
- **Queries** can perform many different functions in a database. Their most common function is to retrieve specific data from the tables.
- The data you want to see is usually spread across several tables, and queries allow you to view it in a single datasheet.
- Also, since you usually don't want to see all the records at once, queries let you add criteria to "filter" the data down to just the records you want.
- Queries come in two basic varieties: **select** queries and **action** queries. A select query simply retrieves the data and makes it available for use.

# Introduction to MS Access

## • Queries



- You can view the results of the query on the screen, print it out, or copy it to the clipboard. Or, you can use the output of the query as the record source for a form or report.
- An action query, as the name implies, performs a task with the data. Action queries can be used to create new tables, add data to existing tables, update data, or delete data.



ERASMUS+

# Introduction to MS Access

- **Macros**



- **Macros** in Access can be thought of as a simplified programming language which you can use to add functionality to your database.
- For example, you can attach a macro to a command button on a form so that the macro runs whenever the button is clicked.
- Macros contain actions that perform tasks, such as opening a report, running a query, or closing the database.
- Most database operations that you do manually can be automated by using macros, so they can be great time-saving devices.



# Introduction to MS Access

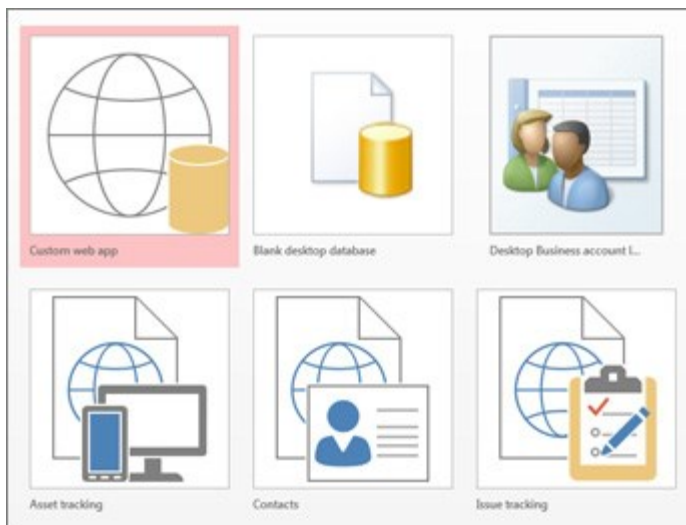
- **Modules**



- Modules, like macros, are objects you can use to add functionality to your database. Whereas you create macros in Access by choosing from a list of macro actions, you write modules in the Visual Basic for Applications (VBA) programming language. A module is a collection of declarations, statements, and procedures that are stored together as a unit.
- A module can be either a class module or a standard module. Class modules are attached to forms or reports, and usually contain procedures that are specific to the form or report they're attached to.
- Standard modules contain general procedures that aren't associated with any other object. Standard modules are listed under Modules in the Navigation Pane, whereas class modules are not.

# Basic tasks for an Access desktop database

- **Choose a template**
  - **Access templates have built-in tables, queries, forms, and reports that are ready to use. A choice of templates is the first thing you'll notice when you start Access, and you can search online for more templates.**



1. In Access click **File** > **New**.

2. Select a desktop database template and enter a name for your database under **File Name**. **(If you don't see a template that would work for you, use the Search online templates box.)**

3. You can either use the default location that Access shows below the **File Name** box or click the folder icon to pick one.

4. Click **Create**.


# Basic tasks for an Access desktop database

- **Choose a template**


Create a contacts database to manage information about people that your team works with, such as customers and partners.

Download size: 355 KB

Rating: ★ ★ ★ ★ ☆ (7154 Votes)

File Name  
 

C:\Users\

  
Create

If Access displays a **Login** dialog box with an empty list of users:

- Click **New User**.
- Fill in the **User Details** form.
- Click **Save & Close**.
- Select the user name you just entered, and then click **Login**.

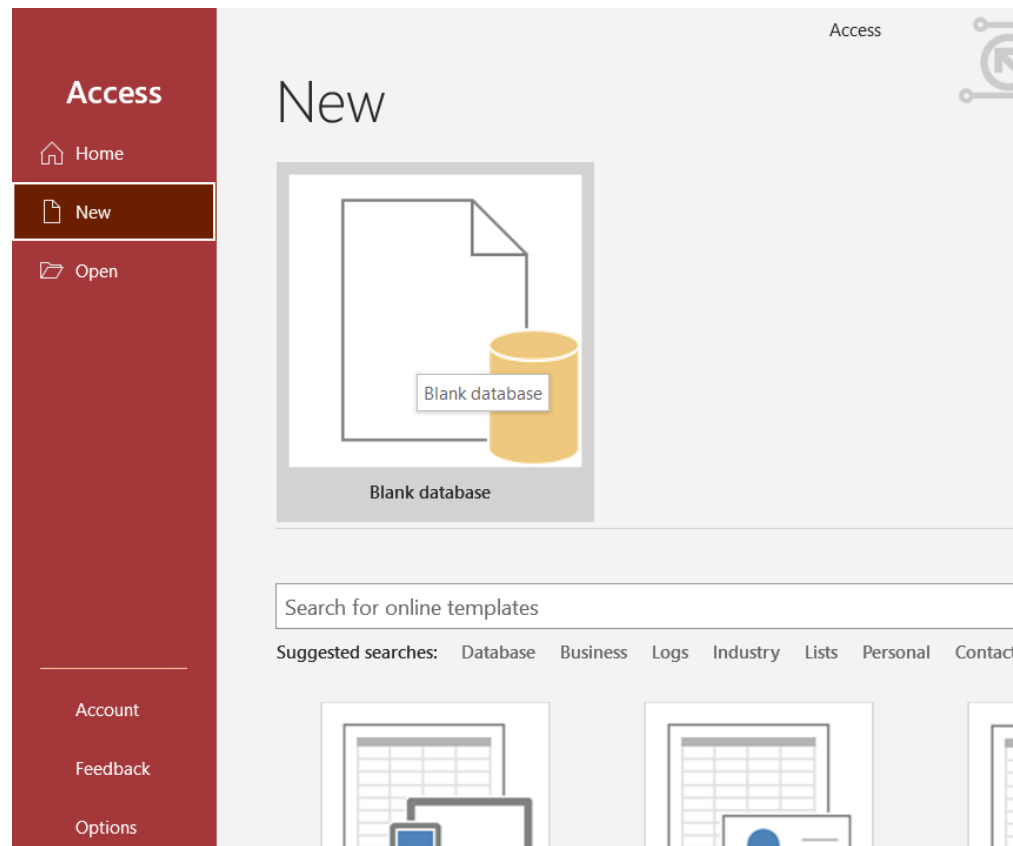
If Access displays a **Security Warning** message in the message bar, and you trust the source of the template, click **Enable Content**. If the database requires a login, log in again.

# Basic tasks for an Access desktop database

- **Create a new database**
  - If none of the templates fit your needs, you might start with a blank desktop database.
    - From Access, click **New > Blank desktop database**.
    - Type a name for your database in the **File Name** box.
    - You can either use the default location that Access shows below the **File Name** box or click the folder icon to pick one.
    - Click **Create**.

# Basic tasks for an Access desktop database

- Create a new database



ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# Basic tasks for an Access desktop database

- **Add a table**
  - **In a database, your information is stored in multiple related tables. To create a table:**
    1. When you open your database for the first time, you'll see a blank table in Datasheet view where you can add data. To add another table, click the **Create** tab > **Table**. You can either start entering data in the empty field (cell) or paste data from another source like an Excel workbook.
    2. To rename a column (field), double-click the column heading, and then type the new name.

# Basic tasks for an Access desktop database

## • Tables

- A relational database like Access usually has several related tables.
- In a well-designed database, each table stores data about a particular subject, such as employees or products.
- A table has records (rows) and fields (columns).
- Fields have different types of data, such as text, numbers, dates, and hyperlinks.

Customers				
	ID	Company	First Name	Last Name
+	1	Company A	Anna	Bedecs
+	2	Company B	Antonio	Gratacos Solsona
+	3	Company C	Thomas	Axen

1.A record: Contains specific data, like information about a particular employee or a product.

2.A field: Contains data about one aspect of the table subject, such as first name or e-mail address.

3.A field value: Each record has a field value. For example, Contoso, Ltd. or someone@example.com.

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices

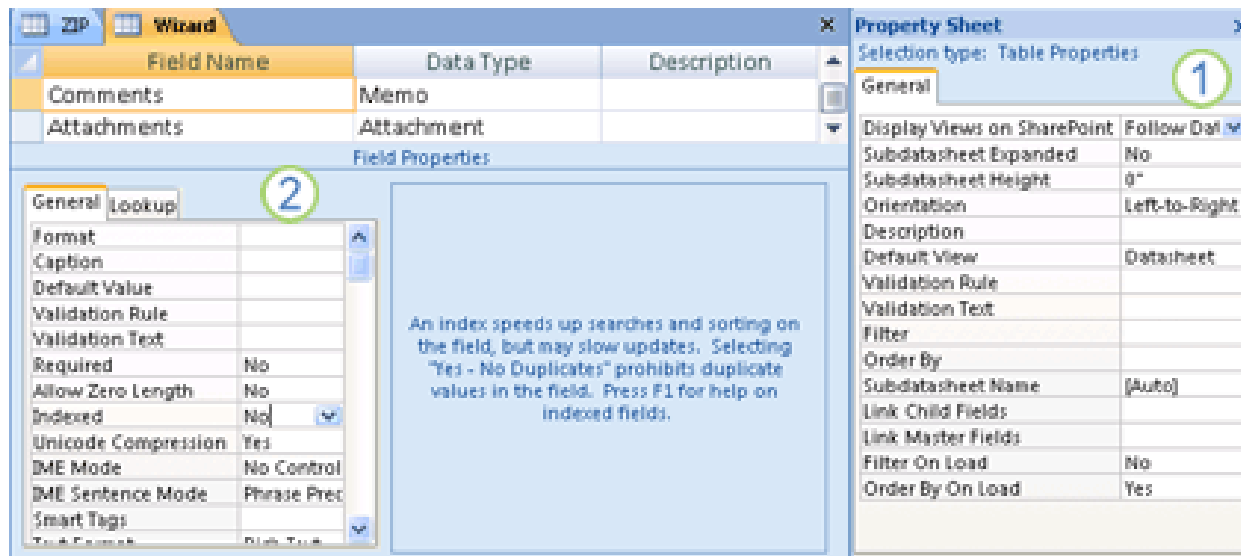
Action Type KA226 - Partnerships for Digital Education Readiness



# Basic tasks for an Access desktop database

## • Tables

- Tables and fields also have properties that you can set to control their characteristics or behavior:



1. Table properties
2. Field properties

In an Access database, table properties are attributes of a table that affect the appearance or behavior of the table as a whole. Table properties are set in the table's property sheet, in Design view.

# Basic tasks for an Access desktop database

- **Tables**

- **Data types.** Every field has a data type. A field's data type indicates the kind of data that the field stores, such as large amounts of text or attached files:

Field Name	Data Type	Description
Comments	Memo	
Attachments	Attachment	
Field Properties		

A data type is a field property, but it differs from other field properties as follows:

- You set a field's data type in the table design grid, not in the **Field Properties** pane.
- A field's data type determines what other properties the field has.
- You must set a field's data type when you create the field.

# Basic tasks for an Access desktop database

- **Table relationships**

- Although each table stores data about a different subject, tables in an Access database usually store data about subjects that are related to each other. For example, a database might contain:
  - A customers table that lists your company's customers and their addresses.
  - A products table that lists the products that you sell, including prices and pictures for each item.
  - An orders table that tracks customer orders.
- Because you store data about different subjects in separate tables, you need some way to tie the data together so that you can easily combine related data from those separate tables. To connect the data stored in different tables, you create relationships. A relationship is a logical connection between two tables that specifies fields that the tables have in common.

# Basic tasks for an Access desktop database

- **Table relationships**

- Fields that are part of a table relationship are called keys. A key usually consists of one field, but may consist of more than one field. There are two kinds of **keys**:
  - **Primary key** - A table can have only one primary key. A primary key consists of one or more fields that uniquely identify each record that you store in the table. Often, there is a unique identification number, such as an ID number, a serial number, or a code, that serves as a primary key. For example, you might have a Customers table where each customer has a unique customer ID number. The customer ID field is the primary key of the Customers table. When a primary key contains more than one field, it is usually composed of pre-existing fields that, taken together, provide unique values. For example, you might use a combination of last name, first name, and birth date as the primary key for a table about people. For more information, see adding or changing a table's primary key.
  - **Foreign key** - A table can also have one or more foreign keys. A foreign key contains values that correspond to values in the primary key of another table. For example, you might have an Orders table in which each order has a customer ID number that corresponds to a record in a Customers table. The customer ID field is a foreign key of the Orders table.

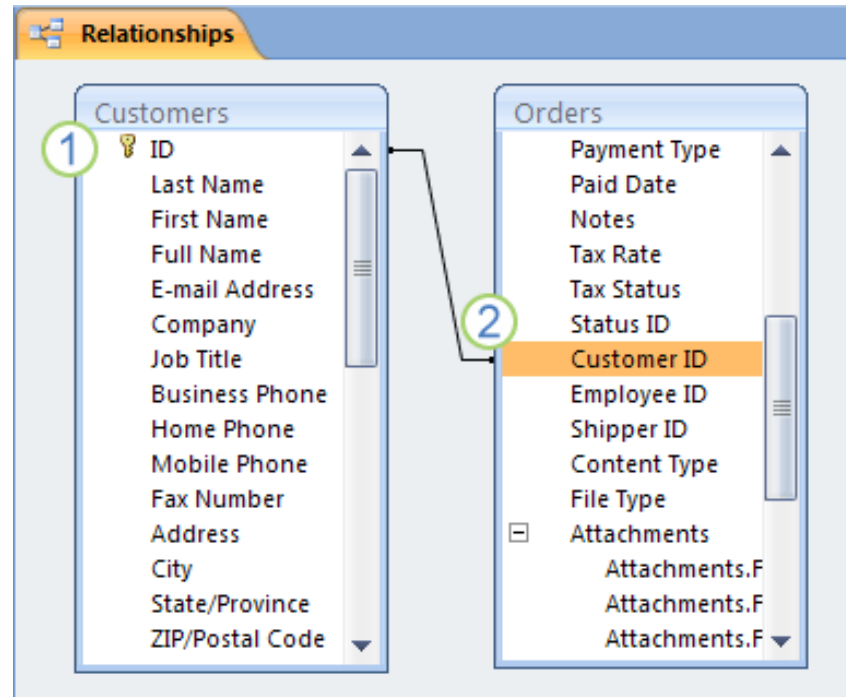
# Basic tasks for an Access desktop database

- **Table relationships**

- The correspondence of values between key fields forms the basis of a table relationship. You use a table relationship to combine data from related tables. For example, suppose that you have a Customers table and an Orders table. In your Customers table, each record is identified by the primary key field, ID.
- To associate each order with a customer, you add a foreign key field to the Orders table that corresponds to the ID field of the Customers table, and then create a relationship between the two keys. When you add a record to the Orders table, you use a value for customer ID that comes from the Customers table. Whenever you want to view any information about an order's customer, you use the relationship to identify which data from the Customers table corresponds to which records in the Orders table.

# Basic tasks for an Access desktop database

- Table relationships



1. A primary key, identified by the key icon next to the field name.
2. A foreign key — note the absence of the key icon.

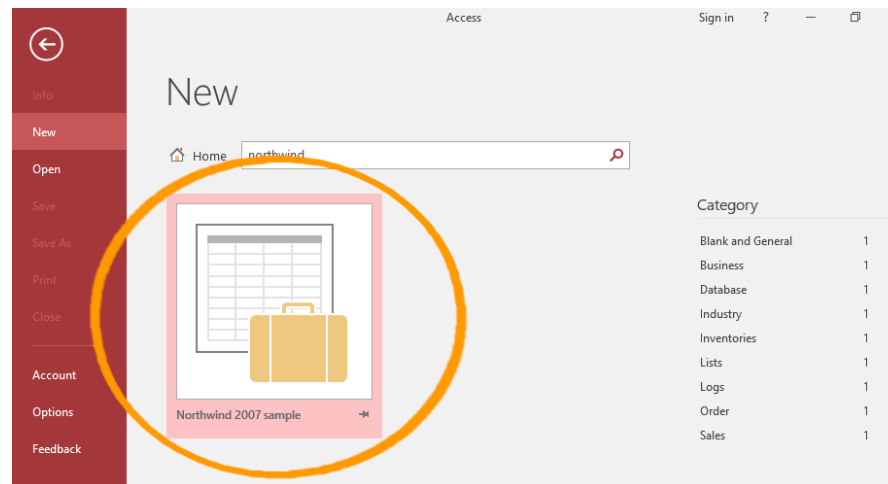
# Basic tasks for an Access desktop database

- **Benefits of using relationships**
  - **Consistency** - Because each item of data is recorded only once, in one table, there is less opportunity for ambiguity or inconsistency. For example, you store a customer's name only once, in a table about customers, rather than storing it repeatedly (and potentially inconsistently) in a table that contains order data.
  - **Efficiency** - Recording data in only one place means you use less disk space. Moreover, smaller tables tend to provide data more quickly than larger tables. Finally, if you don't use separate tables for separate subjects, you will introduce null values (the absence of data) and redundancy into your tables, both of which can waste space and impede performance.
  - **Comprehensibility** - The design of a database is easier to understand if the subjects are properly separated into tables.
- **Plan your tables with relationships in mind**



# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



Using this database, please review the different parts (tables, forms, reports, queries, etc.) of a typical Access database try to create new tables and forms.



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 5. Practice

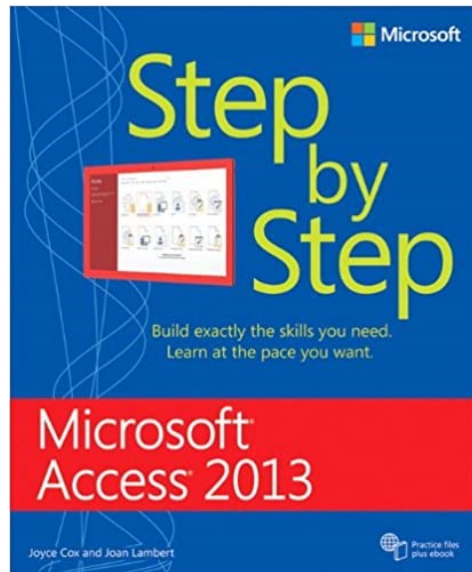
### ☐ Topic 1. Introduction to the computer program MS Access. Use of tables, subforms, filters and reports.

#### ☐ Practical lesson 2. Use of tables and subforms. Using filters and reports.

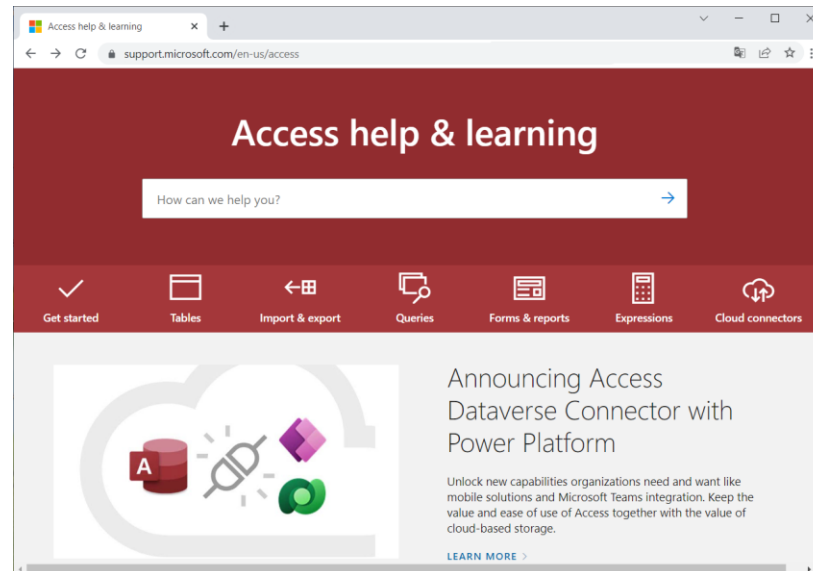


ERASMUS+

# Practical lesson: Use of tables and subforms. Using filters and reports.



*Joyce Cox, Joan Lambert.  
Microsoft Access 2013 Step By  
Step, Microsoft Press., 2013.*



<https://support.microsoft.com/en-us/access>

## Introduction

- When you are working with relational data (related data that is stored in separate tables), you often need to view **multiple tables or queries on the same form**.
- For example, you might want to see customer data from one **table** and information about customer orders from another **table** at the same time.
- **Subforms** are a convenient tool for doing this, and **MS Access** provides several ways to help you create subforms quickly.

## Using tables

- **Tables** are the core database objects. Their purpose is to store information.
- The purpose of every other database object is to interact in some manner with one or more tables. An
- Access database can contain thousands of tables, and the number of records each table can contain is limited more by the storage space available than by anything else.
- Every Access object has two or more views. For tables, the two most common views are **Datasheet view**, in which you can display and modify the table's data, and **Design view**, in which you can display and modify the table's structure.

## Using tables

- To open a table in **Datasheet view**, either double-click its name in the Navigation pane, or right-click its name and then click Open.
- To open a table in **Design view**, right-click its name and then click Design View.
- When a table is open in Datasheet view, clicking the View button in the Views group on the Home tab switches to Design view; when it is open in Design view, clicking the button switches to Datasheet view.
- You can also switch the view by clicking one of the buttons on the View Shortcuts toolbar in the lower-right corner of the program window.



## Using tables

- **Datasheet view** displays the table's data in columns (fields) and rows (records). The first row contains column headings (field names). In this format, the table is often simply referred to as a datasheet.

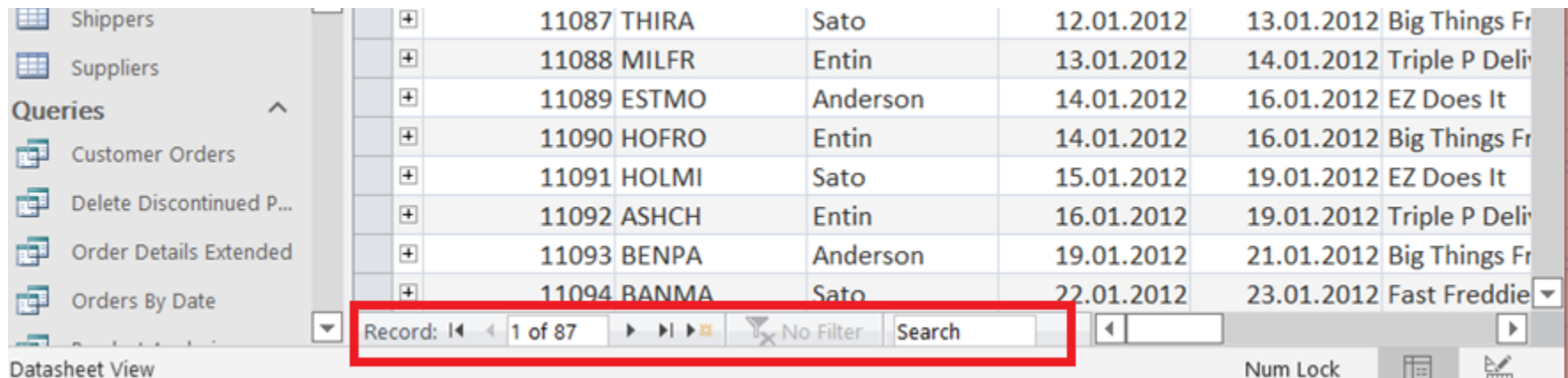
Field names

Categories		
Category I	Category Nam	Description
1	Bulbs	Spring, summer and fall, forced
2	Cacti	Indoor cactus plants
3	Ground covers	Herbaceous perennials, evergreen and deciduous shrubs, ivy, vines, mosses
4	Grasses	Lawn grasses for cool climates
5	Flowers	A wide variety of flowers
6	Wetland plants	Plants suitable for water gardens and bogs
7	Soils/sand	Potting soils, peat moss, mulch, bark
8	Fertilizers	A variety of fertilizers
13	Trees	Evergreen and deciduous trees
14	Herbs	For flavoring and fragrance
15	Bonsai supplies	Bonsai supplies
16	Roses	Many types of roses
17	Rhododendron	Hardy cultivars
18	Pest control	Non-toxic alternatives
19	Carnivorous	Meat-eating plants
20	Tools	Miscellaneous gardening hardware
21	Berry bushes	Small bush fruits
22	Shrubs/hedges	Shrubbery suitable for beds, containers, hedges, etc.
*	(New)	

Field                      Record

## Using tables

- Navigating within tables, adding records and entering data.
  - The bar at the bottom of the table contains several commands to help you search or scroll through records, to add a new record, to save a record and etc.:

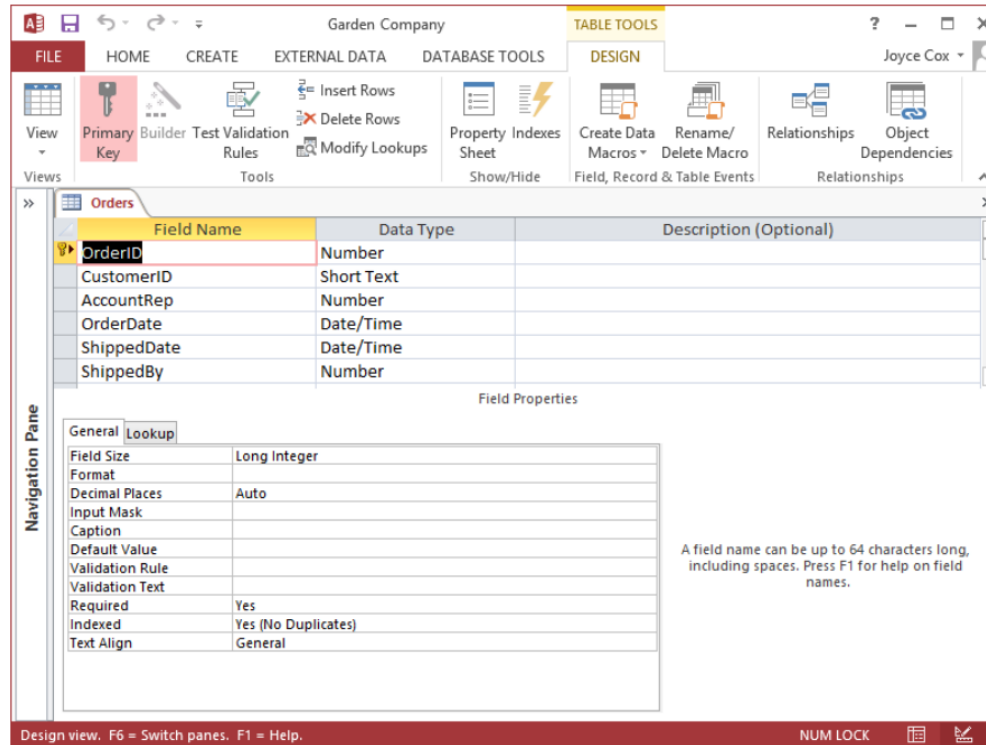


+	11087	THIRA	Sato	12.01.2012	13.01.2012	Big Things Fr	
+	11088	MILFR	Entin	13.01.2012	14.01.2012	Triple P Deliv	
+	11089	ESTMO	Anderson	14.01.2012	16.01.2012	EZ Does It	
+	11090	HOFRO	Entin	14.01.2012	16.01.2012	Big Things Fr	
+	11091	HOLMI	Sato	15.01.2012	19.01.2012	EZ Does It	
+	11092	ASHCH	Entin	16.01.2012	19.01.2012	Triple P Deliv	
+	11093	BENPA	Anderson	19.01.2012	21.01.2012	Big Things Fr	
+	11094	BANMA	Sato	22.01.2012	23.01.2012	Fast Freddie	

Record: 1 of 87    No Filter    Search

## Using tables

- **Design view.** On the View Shortcuts toolbar, click the Design View button to display the according table structure in Design view. Notice that the Design tool tab now appears on the ribbon.



**Datasheet view** displays the data stored in the table, whereas **Design view** displays the underlying table structure.

## About subforms

- A **subform** is a form that is inserted in another form.
- The **primary form** is called the main form, and the form that is enclosed in form is called the **subform**.
- A form/subform combination is sometimes referred to as a hierarchical form, a master/detail form, or a parent/child form.

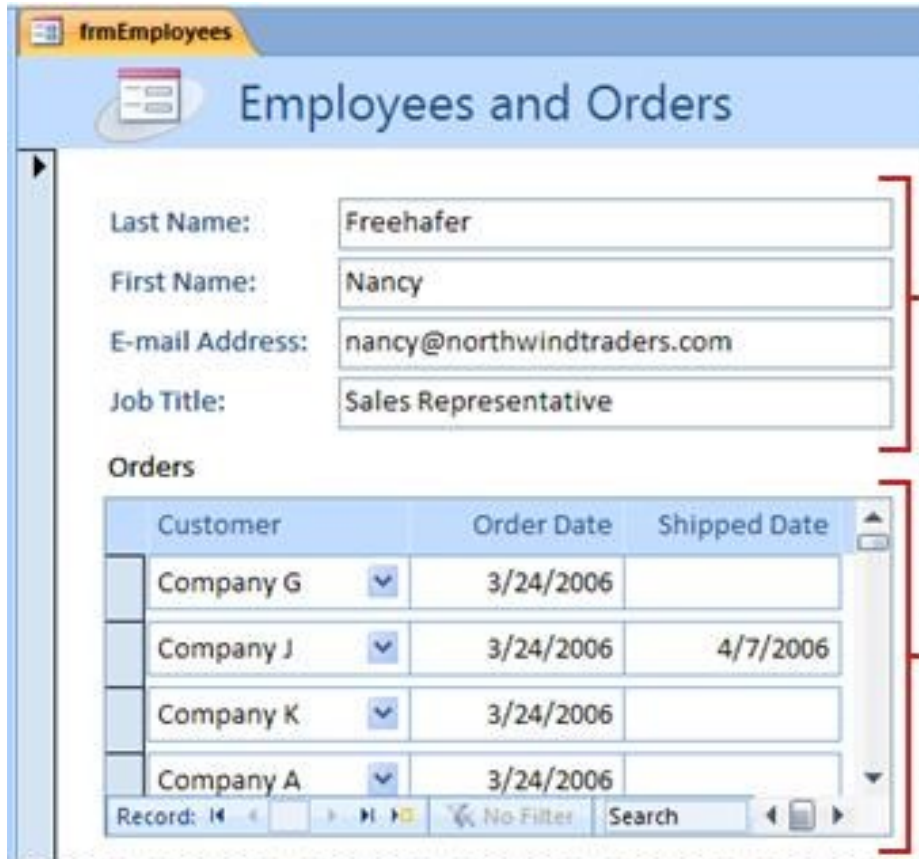
## About subforms

- **Subforms** are especially effective when you want to show data from tables or queries that have a one-to-many relationship.
- A **one-to-many** relationship is an association between two tables in which the **primary key** value of each record in the primary table corresponds to the value in the matching field or fields of **many records** in the related table.

## About subforms

- For example, you can create a form that displays **employee data**, and contains a subform that displays each **employee's orders**.
- The data in the Employees table is the "**one**" side of the relationship.
- The data in the Orders table is the "**many**" side of the relationship — each employee can have more than one order.

# About subforms



The screenshot shows an Access form titled "Employees and Orders". The main form contains fields for "Last Name", "First Name", "E-mail Address", and "Job Title". The "Orders" subform is a table with columns "Customer", "Order Date", and "Shipped Date".

Customer	Order Date	Shipped Date
Company G	3/24/2006	
Company J	3/24/2006	4/7/2006
Company K	3/24/2006	
Company A	3/24/2006	

1. The main form shows data from the **"one"** side of the relationship.

2. The subform shows data from the **"many"** side of the relationship.



## About subforms

- The main form and subform in this kind of form are linked so that the subform displays only records that are related to the current record in the main form.
- For example, when the main form displays Nancy Freehafer's information, the subform displays only her orders.
- If the form and subform were unlinked, the subform would display all the orders, not just Nancy's.

## About subforms

- The following table defines some of the terminology that is associated with subforms:

Term	Definition
Subform control	The control that embeds a form into a form. You can think of the subform control as a "view" of another object in your database, whether it is another form, a table, or a query. The subform control provides properties which allow you to link the data displayed in the control to the data on the main form.
Source Object property	The property of the subform control that determines what object is displayed in the control.

# About subforms

## Datasheet

A simple display of data in rows and columns, much like a spreadsheet. The subform control displays a datasheet when its source object is a table or query, or when its source object is a form whose Default View property is set to Datasheet. In these cases, the subform is sometimes referred to as a datasheet or subdatasheet instead of as a subform.

## Link Child Fields property

The property of the subform control that specifies which field or fields in the subform link the subform to the main form.

## Link Master Fields property

The property of the subform control that specifies which field or fields on the main form link the main form to the subform.

## Create or add a subform

- Use the following table to determine which procedure is most appropriate for your situation:

Scenario	Recommended procedure
You want Access to create both a main form and a subform, and to link the subform to the main form.	Create a form that contains a subform by using the Form Wizard
You want to use an existing form as the main form, but you want Access to create a new subform and add it to the main form.	Add one or more subforms to an existing form by using the Subform Wizard
You want to use an existing form as the main form, and you want to add one or more existing forms to that form as subforms.	Create a subform by dragging one form onto another

# Create a form that contains a subform by using the Form Wizard

- This procedure creates a new form and subform combination by using the Form Wizard:
  1. On the **Create** tab, in the **Forms** group, click **Form Wizard**.
  2. On the first page of the wizard, in the **Tables/Queries** drop-down list, select a table or query. For this example, to create an Employees form that displays orders for each employee in a subform, we will select **Table: Employees** (the "one" side of the one-to-many relationship).

## Create a form that contains a subform by using the Form Wizard

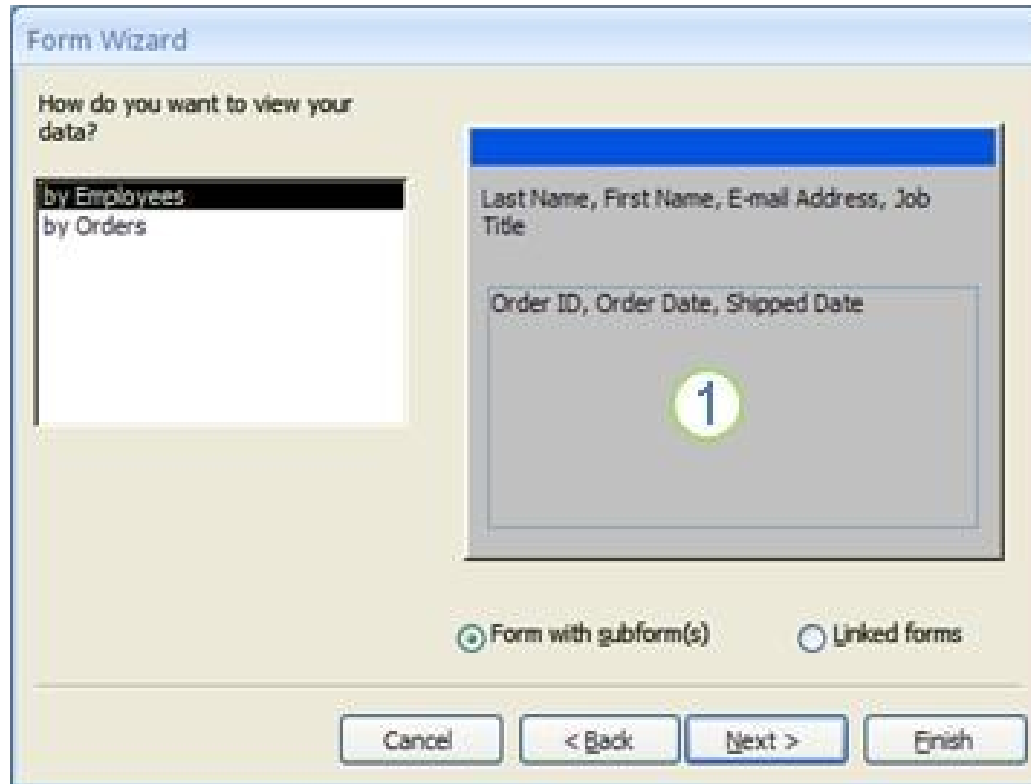
3. Double-click the fields that you want to include from this table or query.
4. On the same page of the wizard, in the **Tables/Queries** drop-down list, select another table or query from the list. For this example, we will select the Orders table (the "many" side of the one-to-many relationship).
5. Double-click the fields that you want to include from this table or query.

## Create a form that contains a subform by using the Form Wizard

6. When you click **Next**, assuming that you set up the relationships correctly before you started the wizard, the wizard asks **How do you want to view your data?** — that is, by which table or query. Select the table on the "one" side of the one-to-many relationship. For this example, to create the Employees form, we will click **by Employees**. The wizard displays a small diagram of a form. The page should resemble the following illustration:



# Create a form that contains a subform by using the Form Wizard



The image shows a 'Form Wizard' dialog box. The title bar says 'Form Wizard'. The main text asks 'How do you want to view your data?'. On the left, there is a list box with two options: 'by Employees' (which is selected and highlighted) and 'by Orders'. On the right, there is a preview area showing a form layout. The top section of the form is labeled 'Last Name, First Name, E-mail Address, Job Title'. Below this is a section labeled 'Order ID, Order Date, Shipped Date'. In the center of this lower section is a large green circle with the number '1' inside it. At the bottom of the dialog, there are two radio buttons: 'Form with subform(s)' (which is selected) and 'Linked forms'. Below the radio buttons are four buttons: 'Cancel', '< Back', 'Next >', and 'Finish'.

**The box in the lower portion of the form  
diagram represents the subform.**

# Create a form that contains a subform by using the Form Wizard

7. At the bottom of the wizard page, select **Form with subform(s)**, and then click **Next**.
8. On the **What layout would you like for your subform?** page, click the layout option that you want, and then click **Next**. Both layout styles arrange the subform data in rows and columns, but a tabular layout is more customizable. You can add color, graphics, and other formatting elements to a tabular subform, whereas a datasheet is more compact, like the datasheet view of a table.
9. On the next page of the wizard, select a formatting style for the form, and then click **Next**. If you chose **Tabular** on the previous page, the formatting style you choose will also be applied to the subform.

# Add one or more subforms to an existing form by using the SubForm Wizard

Use this procedure to add one or more subforms to an existing form. For each subform, you can choose to have Access create a new form or use an existing form as the subform.one for the subform itself:

1. Right-click the existing form in the Navigation Pane, and then click **Design View**.
2. On the **Design** tab, in the **Controls** group, click the down-arrow to display the **Controls** gallery, and ensure that **Use Control Wizards** is selected.
3. On the **Design** tab, in the **Controls** group, click the **Subform/Subreport** button.
4. Click on the form where you want to place the subform.
5. Follow the directions in the wizard.
6. When you click **Finish**, Access adds a subform control to your form. If you chose to have Access create a new form for the subform instead of using an existing form, Access creates the new form object and adds it to the Navigation Pane.

# Create a subform by dragging one form onto another

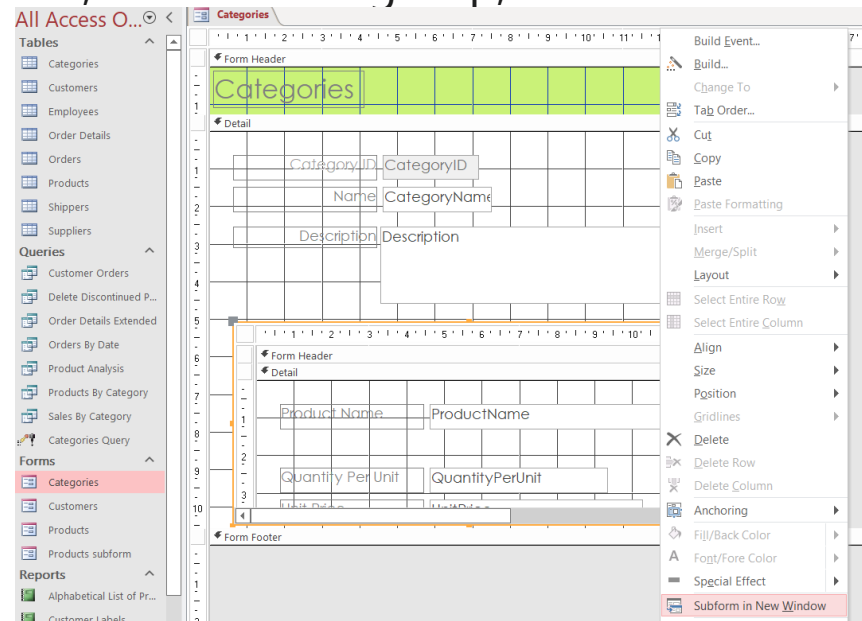
Use this procedure if you want to use an existing form as a main form, and you want to add one or more existing forms to that form as subforms:

1. In the Navigation Pane, right-click the form that you want to use as the main form, and then click **Layout View**.
2. Drag the form that you want to use as the subform from the Navigation Pane onto the main form.
3. Access adds a subform control to the main form and binds the control to the form that you dragged from the Navigation Pane. Access also tries to link the subform to the main form, based on the relationships that have been defined in your database.
4. Repeat this step to add any additional subforms to the main form.
5. To verify that the linking was successful, on the **Home** tab, in the **Views** group, click **View**, click **Form View**, and then use the main form's record selector to advance through several records. If the subform filters itself correctly for each employee, then the procedure is complete.

# Open a subform in a new window in Design view

If you want to make design changes to a subform while you are working on its main form in Design view, you can open the subform in its own window:

1. Click the subform to select it.
2. On the **Design** tab, in the **Tools** group, click **Subform in New Window**.



# Change the default view of a subform

When you add a subform to a form, the **subform/subreport** control displays the subform according to the subform's **Default View** property. This property can be set to the following values:

- Single Form
- Continuous Forms
- Datasheet
- Split Form

When you first create a subform, this property may be set to **Continuous Forms** or perhaps **Single Form**. However, if you set the **Default View** property of a subform to **Datasheet**, then the subform will display as a datasheet on the main form.

# Change the default view of a subform

To set the **Default View** property of a subform:

1. Close any open objects.
2. In the Navigation Pane, right-click the subform and then click **Design View**.
3. If the Property Sheet is not already displayed, press F4 to display it.
4. In the drop-down list at the top of the Property Sheet, make sure **Form** is selected.
5. On the **Format** tab of the Property Sheet, set the **Default View** property to the view you want to use.



## Reports in MS Access

- Reports offer a way to view, format, and summarize the information in your Microsoft Access database.
- For example, you can create a simple report of phone numbers for all your contacts, or a summary report on the total sales across different regions and time periods.
- What can you do with a report?

A report is a database object that comes in handy when you want to present the information in your database for any of the following uses:

- Display or distribute a summary of data.
- Archive snapshots of the data.
- Provide details about individual records.
- Create labels.

# Reports in MS Access

## Create a report in Access:

### Step 1: Choose a record source

- The record source of a report can be a table, a named query, or an embedded query. The record source must contain all of the rows and columns of data you want display on the report.
  - If the data is from an existing table or query, select the table or query in the Navigation Pane, and then continue to Step 2.
  - If the record source does not yet exist, do one of the following:
    - Continue to [Step 2](#) and use the **Blank Report** tool,
    - Create the table(s) or query that contains the required data. Select the query or table in the Navigation Pane, and then continue to [Step 2](#).

# Reports in MS Access

## Create a report in Access:

### Step 2: Choose a report tool

- The report tools are located on the Create tab of the ribbon, in the Reports group. The following table describes the options:

Tool	Description
Report	Creates a simple, tabular report containing all of the fields in the record source you selected in the Navigation Pane.
Report Design	Opens a blank report in Design view, to which you can add the required fields and controls.
Blank Report	Opens a blank report in Layout view, and displays the Field List from where you can add fields to the report
Report Wizard	Displays a multiple-step wizard that lets you specify fields, grouping/sorting levels, and layout options.
Labels	Displays a wizard that lets you select standard or custom label sizes, as well as which fields you want to display, and how you want them sorted.

# Reports in MS Access

## Create a report in Access:

### Step 3: Create the report

1. Click the button for the tool you want to use. If a wizard appears, follow the steps in the wizard and click **Finish** on the last page.  
Access displays the report in Layout view.
2. Format the report to achieve the looks that you want:
  - Resize fields and labels by selecting them and then dragging the edges until they are the size you want.
  - Move a field by selecting it (and its label, if present), and then dragging it to the new location.
  - Right-click a field and use the commands on the shortcut menu to merge or split cells, delete or select fields, and perform other formatting tasks.

# Reports in MS Access

## Alphabetical List of Products

### Alphabetical List of Products

A	Product Name	Quantity Per Unit	Units In Stock
	Ambrosia	6 - 2" pots	16
	American Pitcher Plant	1 ea.	4
	Anacharis	1 ea.	2
	Anemone	One dozen	26
	Animal repellent	1 qt.	3
	Animal trap	1 ea.	2
	Anise	6 - 2" pots	20
	Austrian Copper	Per plant	7
	Austrian Pine	One gal. container	10
	Autumn Crocus	One dozen	37
B	Product Name	Quantity Per Unit	Units In Stock
	Baby's Breath	1 ea.	23
	Bat box	1 box per kit	12
	Beautybush	1 ea.	2
	Beebalm	1 ea.	14
	Begonias	6 per pkg.	12
	Beneficial nematodes	1 pt	4
	Blackberries	8 starts per pkg	18
	Bladderwort	One 3" starter	5

## Example reports in MS Access

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

## Filter data in a report

- When you view an Access report on the screen, you can apply filters to zero in on the data you want to see.
- To filter data in a report, open it in Report view (right-click it in the Navigation pane and click **Report View**). Then, right-click the data you want to filter.
- For example, in a report listing all employees, you might want to limit the report to employees whose last names start with "L":
  1. Right-click any last name, and click **Text Filters > Begins With**.

# Filter data in a report

EmployeeHireDates

## Employee Hire Dates

ID	Last Name	First Name	Hire Date
1	Bedecs	Anna	3/27/2013
2	Gratacos Solsona	Antonia	2/5/2013
3	A		3/27/2013
4	L		2/5/2013
5	O		3/27/2013
6	P		12/13/2012
7	X		12/15/2012
8	A		12/1/2012
9	M		3/27/2013
10	V		
11	K		
12	E		
13	L		
14	G		
15	K		
16	Goldenboudt	Daniel	

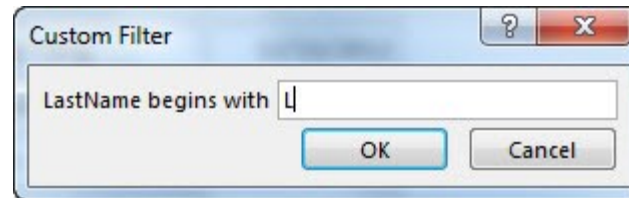
Context menu for row 2 (Gratacos Solsona):

- Cut
- Copy
- Paste
- Insert
- Merge/Split
- Sort A to Z
- Sort Z to A
- Clear filter from LastName
- Text Filters**
  - Equals...
  - Does Not Equal...
  - Begins With...**
  - Does Not Begin With...
  - Contains...
  - Does Not Contain...
  - Ends With...
  - Does Not End With...
- Change To
- Report Properties



## Filter data in a report

- Enter "L" in the box that appears, and click **OK**.



A screenshot of a 'Custom Filter' dialog box. It has a title bar with a question mark and a close button. The main area contains the text 'LastName begins with' followed by a text input field containing the letter 'L'. Below the input field are two buttons: 'OK' and 'Cancel'.

- Access applies the filter, and now you can print the report with just that data.



A screenshot of a report titled 'Employee Hire Dates'. The report displays a table with four columns: ID, Last Name, First Name, and Hire Date. The data is filtered to show only employees whose last name starts with 'L'. There are four rows of data. Below the table, there is a small box containing the number '4', likely representing the total number of records displayed.

ID	Last Name	First Name	Hire Date
4	Lee	Christina	2/5
13	Ludick	Andre	4/1
20	Li	George	5/2
26	Liu	Run	3/27

ERASMUS+

## Filter data in a report

- **Toggle or clear filters**
  - On the **Home** tab, click the **Toggle Filter** button to remove and reapply the filter as needed.
  - If you close the report without explicitly clearing the filters, Access remembers them and you can click **Toggle Filter** again to reapply them next time you open the report. This works even if you close and reopen the database. However, if you click **Home > Advanced > Clear All Filters**, Access clears the filters completely and you'll need to start from scratch next time around.

## Filter data in a report

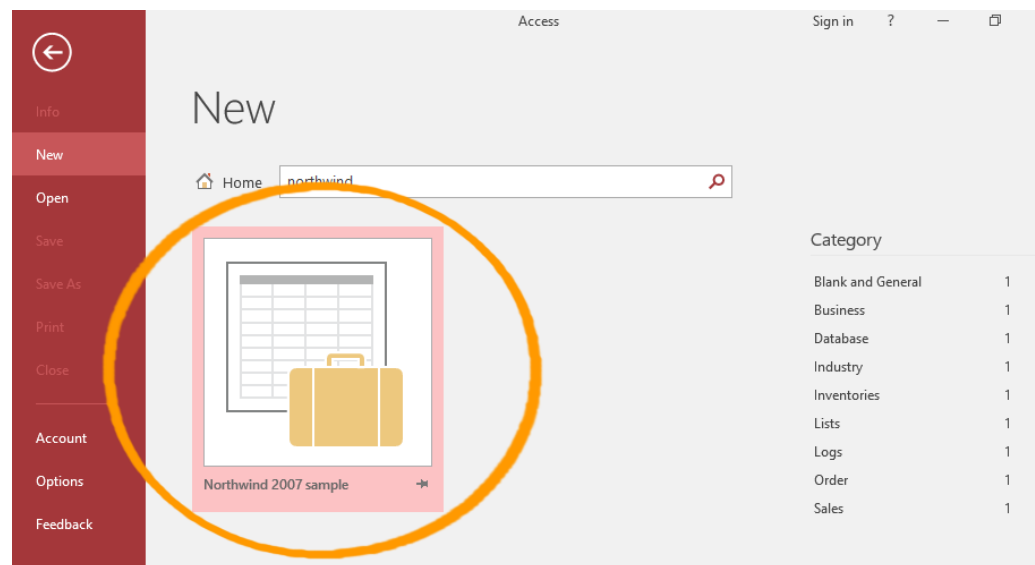
- **Save filters as a query**
  - If you have a lot of filters applied to a report, you might want to save the filters as a query. Then you can use the query as the data source for the current report or a new report, or just run the query next time you want to see the data.
1. Apply the filters, and click **Home** > **Advanced** > **Advanced Filter/Sort**.

Access creates a query that includes all the filters you've applied. If you want to see other columns besides the filtered columns in the query output, double-click them in the tables to add them to the query grid.

2. Click **Save**, and enter a name for the query.

# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



**Test everything from this practical lesson with this database.**



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 5. Practice

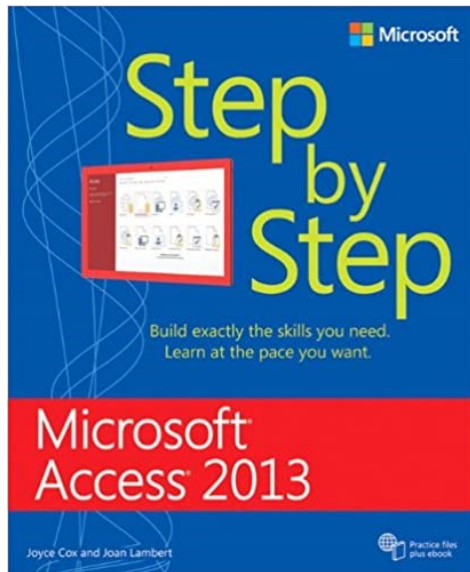
### ☐ Topic 2. Maintaining database changes. Ensuring the reliability of information in the database.

#### ☐ Practical lesson 1. Maintaining database changes

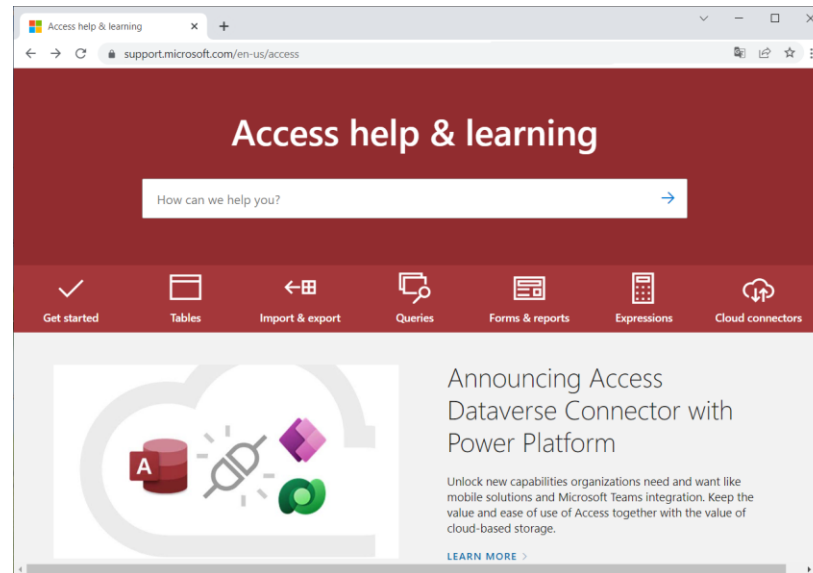


ERASMUS+

# Practical lesson: Maintaining database changes.



*Joyce Cox, Joan Lambert.  
Microsoft Access 2013 Step By  
Step, Microsoft Press., 2013.*



<https://support.microsoft.com/en-us/access>



# Introduction

- A properly **designed database** provides you with access to up-to-date, accurate information.
- Because a **correct design** is essential to achieving your goals in working with a database, investing the time required to learn the principles of good design makes sense.
- In the end, you are much more likely to end up with a database that meets your needs and can easily accommodate **change**.

# What is good database design?

- Certain principles guide the database design process. The first principle is that duplicate information (also called redundant data) is bad, because it wastes space and increases the likelihood of errors and inconsistencies.
- The second principle is that the correctness and completeness of information is important. If your database contains incorrect information, any reports that pull information from the database will also contain incorrect information.
- As a result, any decisions you make that are based on those reports will then be misinformed.

# What is good database design?

- A **good database design** is, therefore, one that:
  - Divides your information into subject-based tables to reduce redundant data.
  - Provides Access with the information it requires to join the information in the tables together as needed.
  - Helps support and ensure the accuracy and integrity of your information.
  - Accommodates your data processing and reporting needs.

# The design process

- The design process consists of the following steps:

- **Determine the purpose of your database**

This helps prepare you for the remaining steps.

- **Find and organize the information required**

Gather all of the types of information you might want to record in the database, such as product name and order number.

- **Divide the information into tables**

Divide your information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.

- **Turn information items into columns**

Decide what information you want to store in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.

# The design process

- The design process consists of the following steps:

- **Specify primary keys**

Choose each table's primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID..

- **Set up the table relationships**

Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.

- **Refine your design**

Analyze your design for errors. Create the tables and add a few records of sample data. See if you can get the results you want from your tables. Make adjustments to the design, as needed.

- **Apply the normalization rules**

Apply the data normalization rules to see if your tables are structured correctly. Make adjustments to the tables, as needed.

## Determining the purpose of your database

- It is a good idea to write down the **purpose** of the database on paper — its purpose, how you expect to use it, and who will use it.
- For a small database for a home based business, for example, you might write something simple like "The customer database keeps a list of customer information for the purpose of producing mailings and reports."
- If the database is more complex or is used by many people, as often occurs in a corporate setting, the purpose could easily be a paragraph or more and should include when and how each person will use the database.
- The idea is to have a well developed mission statement that can be referred to throughout the design process. Having such a statement helps you focus on your goals when you make decisions.

## Finding and organizing the required information

- To find and organize the information required, start with your existing information. For example, you might record purchase orders in a ledger or keep customer information on paper forms in a file cabinet.
- Gather those documents and list each type of information shown (for example, each box that you fill in on a form). If you don't have any existing forms, imagine instead that you have to design a form to record the customer information. What information would you put on the form?
- What fill-in boxes would you create? Identify and list each of these items. For example, suppose you currently keep the customer list on index cards. Examining these cards might show that each card holds a customer's name, address, city, state, postal code and telephone number. Each of these items represents a potential column in a table.



## Finding and organizing the required information

- As you prepare this list, don't worry about getting it perfect at first. Instead, list each item that comes to mind. If someone else will be using the database, ask for their ideas, too. You can fine-tune the list later.
- Next, consider the types of reports or mailings you might want to produce from the database. For instance, you might want a product sales report to show sales by region, or an inventory summary report that shows product inventory levels.
- You might also want to generate form letters to send to customers that announces a sale event or offers a premium. Design the report in your mind, and imagine what it would look like. What information would you place on the report? List each item. Do the same for the form letter and for any other report you anticipate creating.

# Finding and organizing the required information

An illustration of a person sitting at a desk with a laptop, with a thought bubble above them containing a table. The background is a warm orange gradient.

Northwind Traders Product Inventory		
<u>Product Id</u>	<u>Name</u>	<u>Quantity on Hand</u>
1	Chai	36
2	Chang	17

- Giving thought to the reports and mailings you might want to create helps you identify items you will need in your database. For example, suppose you give customers the opportunity to opt in to (or out of) periodic e-mail updates, and you want to print a listing of those who have opted in. To record that information, you add a “Send e-mail” column to the customer table. For each customer, you can set the field to Yes or No.

# Dividing the information into tables

- To divide the information into tables, choose the major entities, or subjects. For example, after finding and organizing information for a product sales database, the preliminary list might look like this:

A hand-drawn diagram on a piece of paper with two binder holes, showing a preliminary database schema. It lists four entities: Customers, Suppliers, Products, and Orders, each with its associated attributes.

<b>Customers</b> Name Address City, State, Zip Send E-mail Salutation E-mail Address	<b>Products</b> Product Name Price Units in Stock Units on Order
<b>Suppliers</b> Company Name Contact Name Address City, State, Zip	<b>Orders</b> Order Number Salesperson Order Date Product Quantity Price Total

The major entities shown here are the products, the suppliers, the customers, and the orders. Therefore, it makes sense to start out with these four tables: one for facts about products, one for facts about suppliers, one for facts about customers, and one for facts about orders.

# Dividing the information into tables

- When you first review the preliminary list of items, you might be tempted to place them all in a **single table**, instead of the four shown in the preceding illustration. You will learn here why that is a **bad idea**. Consider for a moment, the table shown here:

ProductName	Suppliers	Address
Chai	Exotic Liquids	49 Gilbert St.
Chang	Exotic Liquids	49 Gilbert St.
Aniseed Syrup	Exotic Liquids	49 Gilbert St.
Chef Anton's Cajun Seafood	New Orleans Cajun Deli	P.O. Box 78934

- In this case, each row contains information about both the product and its supplier. Because you can have many products from the same supplier, the supplier name and address information has to be repeated many times. This wastes disk space. Recording the supplier information only once in a separate Suppliers table, and then linking that table to the Products table, is a much better solution.

# Dividing the information into tables

- A **second problem** with this design comes about when you need to **modify** information about the supplier. For example, suppose you need to **change** a supplier's address. Because it appears in many places, you might accidentally **change** the address in one place but forget to **change** it in the others. Recording the supplier's address in only one place solves the problem.
- When you design your database, always try to record each fact **just once**. If you find yourself repeating the same information in more than one place, such as the address for a particular supplier, place that information in a **separate table**.
- Once you have chosen the subject that is represented by a table, columns in that table should store facts **only about the subject**. For instance, the product table should store facts only about products. Because the supplier address is a fact about the supplier, and not a fact about the product, it belongs in the supplier table.

# Turning information items into columns

- To determine the columns in a table, decide what information you need to track about the subject recorded in the table. For example, for the Customers table, Name, Address, City-State-Zip, Send e-mail, Salutation and E-mail address comprise a good starting list of columns.
- Each record in the table contains the same set of columns, so you can store Name, Address, City-State-Zip, Send e-mail, Salutation and E-mail address information for each record.
- For example, the address column contains customers' addresses. Each record contains data about one customer, and the address field contains the address for that customer.
- Once you have determined the initial set of columns for each table, you can further refine the columns.



## Turning information items into columns

- For example, it makes sense to store the customer name as two separate columns: first name and last name, so that you can sort, search, and index on just those columns.
- Similarly, the address actually consists of five separate components, address, city, state, postal code, and country/region, and it also makes sense to store them in separate columns.
- If you want to perform a search, filter or sort operation by state, for example, you need the state information stored in a separate column.
- You should also consider whether the database will hold information that is of domestic origin only, or international, as well. For instance, if you plan to store international addresses, it is better to have a Region column instead of State, because such a column can accommodate both domestic states and the regions of other countries/regions.



## Specifying primary keys

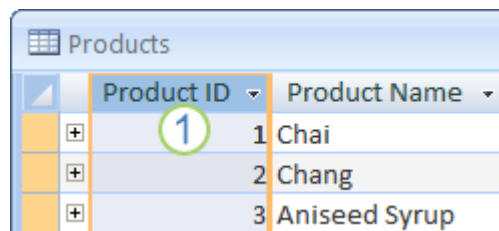
- Each table should include a column or set of columns that **uniquely** identifies each row stored in the table.
- This is often a unique identification number, such as an employee ID number or a serial number. In database terminology, this information is called the primary key of the table.
- Access uses **primary key** fields to quickly associate data from multiple tables and bring the data together for you.
- You cannot have **duplicate** values in a primary key. For example, don't use people's names as a primary key, because names are not unique. You could easily have two people with the same name in the same table.
- A primary key must always have a value. If a column's value can become unassigned or unknown (a missing value) at some point, it can't be used as a component in a primary key.

## Specifying primary keys

- You should always choose a primary key whose value will **not change**. In a database that uses more than one table, a table's primary key can be used as a reference in other tables.
- If the primary key changes, the change must also be applied everywhere the key is referenced.
- Using a primary key that will not change reduces the chance that the primary key might become out of sync with other tables that reference it.
- Often, an arbitrary unique number is used as the primary key. For example, you might assign each order a unique order number. The order number's only purpose is to identify an order. Once assigned, it never changes.
- If you don't have in mind a column or set of columns that might make a good primary key, consider using a column that has the **AutoNumber** data type

## Specifying primary keys

- When you use the **AutoNumber** data type, Access automatically assigns a value for you. Such an identifier is factless; it contains no factual information describing the row that it represents.
- Factless identifiers are ideal for use as a primary key because they do not change.
- A primary key that contains facts about a row — a telephone number or a customer name, for example — is more likely to change, because the factual information itself might change.

A screenshot of a Microsoft Access table named 'Products'. The table has two columns: 'Product ID' and 'Product Name'. The 'Product ID' column is highlighted with a green circle around the number '1' in the first row, indicating it is the primary key. The rows contain the following data: (1, Chai), (2, Chang), and (3, Aniseed Syrup).

Product ID	Product Name
1	Chai
2	Chang
3	Aniseed Syrup

**1** A column set to the AutoNumber data type often makes a good primary key. No two product IDs are the same.

## Specifying primary keys

- For the product sales database, you can create an AutoNumber column for each of the tables to serve as primary key: ProductID for the Products table, OrderID for the Orders table, CustomerID for the Customers table, and SupplierID for the Suppliers table.

Customers	Products
CustomerID	ProductID
Name	Product Name
Address	Unit Price
City	Units in Stock
Region	Units on Order
Postal Code	Quantity per Unit
Country	
Send E-mail	Orders
Salutation	OrderID
E-mail address	Salesperson
	Order Date
Suppliers	Product
SupplierID	Quantity
Company Name	Price
Contact Name	
Address	
City	
Region	
Postal Code	
Country	
Phone	

ERASMUS+

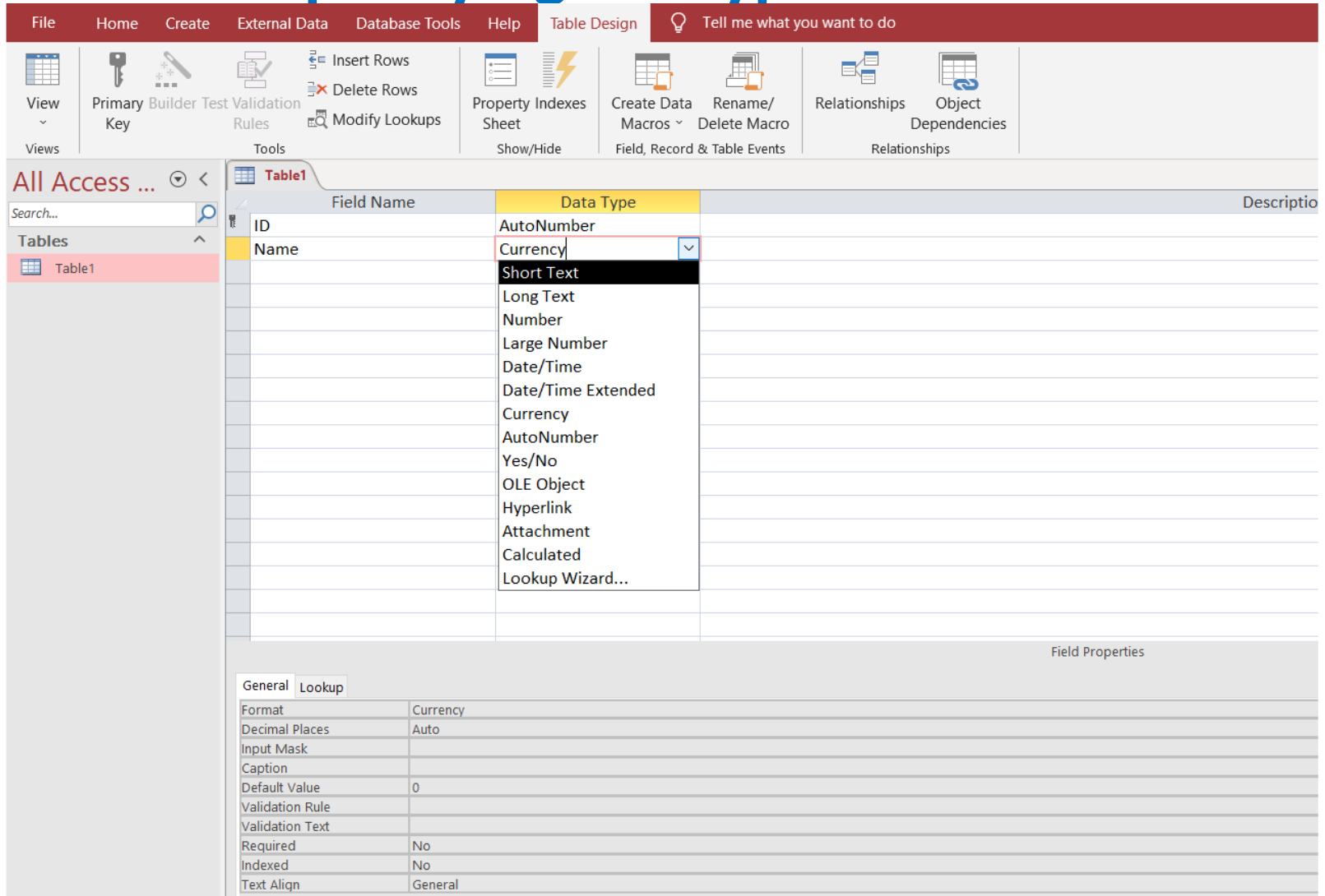
## Specifying the type of data

- The data type restricts entries in that field to a **specific type of data**. For example, if the data type is set to Number and you try to enter text, Access refuses the entry and displays a warning.
- When setting the data type of a field in a table in Design view, you can choose from the following types:
  - **Short Text:** Use for text fields that require up to 255 alphanumeric characters.
  - **Long Text:** Use for text fields that require up to 65,535 alphanumeric characters.
  - **Number:** Use for numeric values. The size of the entry is controlled by the Field Size
  - Property.
  - **Date/Time:** Use for dates in the years from 100 through 9999. Dates and times can be expressed in a variety of formats.
  - **Currency:** Use for decimal values with up to 15 digits to the left of the decimal point and up to 4 digits to the right.

## Specifying the type of data

- When setting the data type of a field in a table in Design view, you can choose from the following types:
  - **AutoNumber:** Use when you want Access to assign a unique number to each new record. If you delete a record, its AutoNumber value is not reused, and remaining records are not updated.
  - **Yes/No:** Use for fields that can have only two possible mutually exclusive values, such as True or False.
  - **OLE Object:** Use to hold a graphic or object such as a Microsoft Excel worksheet or Microsoft Word document.
  - **Hyperlink:** Use to hold a clickable path to a folder on your hard disk, a network location, or a website.
  - **Attachment:** Use to attach a file to a record in the same way that you might attach a file to an email message.
  - **Calculated:** Use to hold the results of a calculation based on other fields in the same table.

# Specifying the type of data



The screenshot shows the Microsoft Access Table Design view for a table named 'Table1'. The ribbon includes 'File', 'Home', 'Create', 'External Data', 'Database Tools', 'Help', and 'Table Design'. The 'Table Design' ribbon has several groups: 'Views' (View, Primary Key), 'Tools' (Builder, Test Validation Rules, Insert Rows, Delete Rows, Modify Lookups), 'Property Indexes Sheet' (Property Indexes, Show/Hide), 'Create Data Macros' (Create Data Macros, Field, Record & Table Events), 'Rename/Delete Macro' (Rename/Delete Macro), 'Relationships' (Relationships, Object Dependencies), and 'Tell me what you want to do'.

The table design grid shows two fields: 'ID' with data type 'AutoNumber' and 'Name' with data type 'Currency'. A dropdown menu is open for the 'Name' field, showing the following options: AutoNumber, Currency, Short Text, Long Text, Number, Large Number, Date/Time, Date/Time Extended, Currency, AutoNumber, Yes/No, OLE Object, Hyperlink, Attachment, Calculated, and Lookup Wizard...

The 'Field Properties' pane is visible at the bottom right, showing the 'Lookup' tab. The 'Format' property is set to 'Currency', and the 'Decimal Places' property is set to 'Auto'.

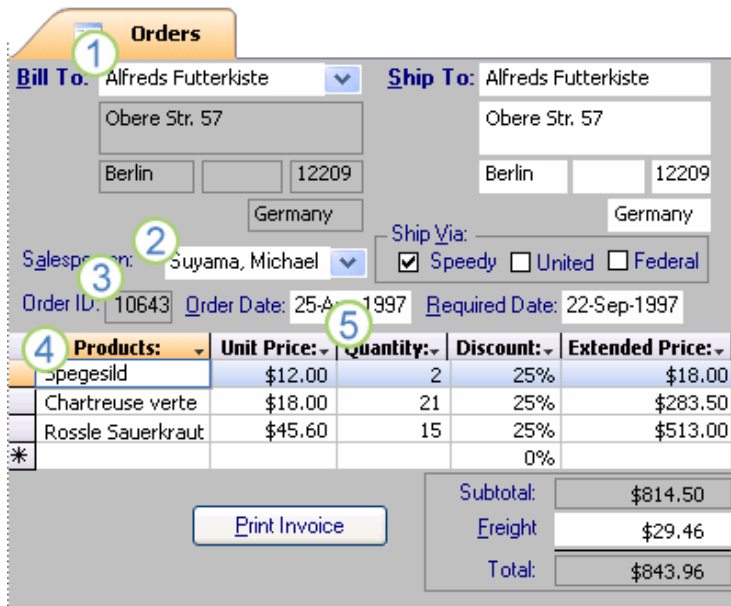
Field Name	Data Type	Description
ID	AutoNumber	
Name	Currency	

Field Properties	
General	Lookup
Format	Currency
Decimal Places	Auto
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	No
Indexed	No
Text Align	General



## Creating the table relationships

- Now that you have divided your information into tables, you need a way to bring the information together again in meaningful ways. For example, the following form includes information from several tables.



**Orders**

**Bill To:** Alfreds Futterkiste  
Obere Str. 57  
Berlin 12209  
Germany

**Ship To:** Alfreds Futterkiste  
Obere Str. 57  
Berlin 12209  
Germany

**Salesperson:** Suyama, Michael

**Ship Via:** ☒ Speedy ☐ United ☐ Federal

**Order ID:** 10643 **Order Date:** 25-Aug-1997 **Required Date:** 22-Sep-1997

Products	Unit Price	Quantity	Discount	Extended Price
Spegesild	\$12.00	2	25%	\$18.00
Chartreuse verte	\$18.00	21	25%	\$283.50
Rossle Sauerkraut	\$45.60	15	25%	\$513.00
*			0%	

**Subtotal:** \$814.50  
**Freight:** \$29.46  
**Total:** \$843.96

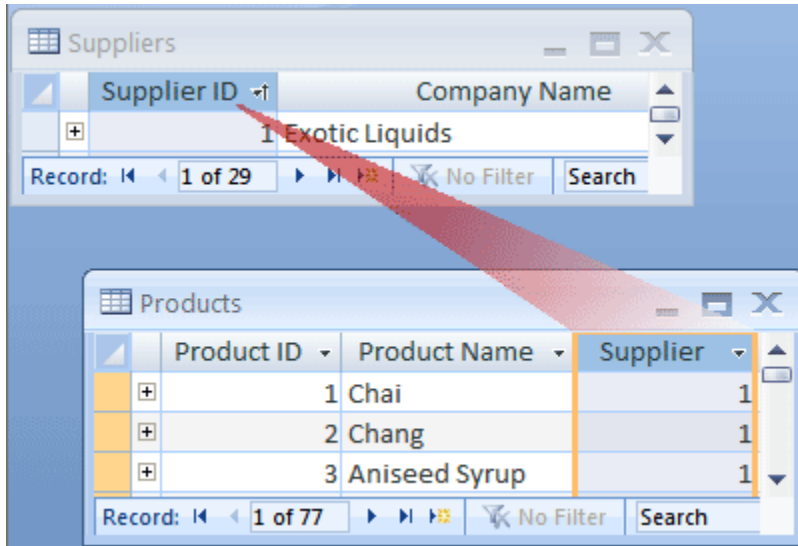
[Print Invoice](#)

1. Information in this form comes from the Customers table...
2. ...the Employees table...
3. ...the Orders table...
4. ...the Products table...
5. ...and the Order Details table.

*Access is a relational database management system. In a relational database, you divide your information into separate, subject-based tables. You then use table relationships to bring the information together as needed.*

## Creating a one-to-many relationship

- Consider this example: the Suppliers and Products tables in the product orders database. A supplier can supply any number of products. It follows that for any supplier represented in the Suppliers table, there can be many products represented in the Products table. The relationship between the Suppliers table and the Products table is, therefore, a **one-to-many** relationship.



The screenshot shows two tables in Microsoft Access:

- Suppliers Table:**

Supplier ID	Company Name
1	Exotic Liquids
- Products Table:**

Product ID	Product Name	Supplier
1	Chai	1
2	Chang	1
3	Aniseed Syrup	1

A red arrow points from the 'Supplier ID' field in the Suppliers table to the 'Supplier' field in the Products table, illustrating the one-to-many relationship.

To represent a one-to-many relationship in your database design, take the primary key on the "**one**" side of the relationship and add it as an additional column or columns to the table on the "**many**" side of the relationship. In this case, for example, you add the Supplier ID column from the Suppliers table to the Products table. Access can then use the supplier ID number in the Products table to locate the correct supplier for each product.

## Creating a one-to-many relationship

- The Supplier ID column in the Products table is called a foreign key. A foreign key is another table's primary key. The Supplier ID column in the Products table is a foreign key because it is also the primary key in the Suppliers table.

Customers	Products
CustomerID	ProductID
Name	Product Name
Address	Unit Price
City	Units in Stock
Region	Units on Order
Postal Code	Quantity per Unit
Country	SupplierID
Send E-mail	
Salutation	Orders
E-mail address	OrderID
	Salesperson
	Order Date
Suppliers	Product
SupplierID	Quantity
Company Name	Price
Contact Name	
Address	
City	
Region	
Postal Code	
Country	
Phone	

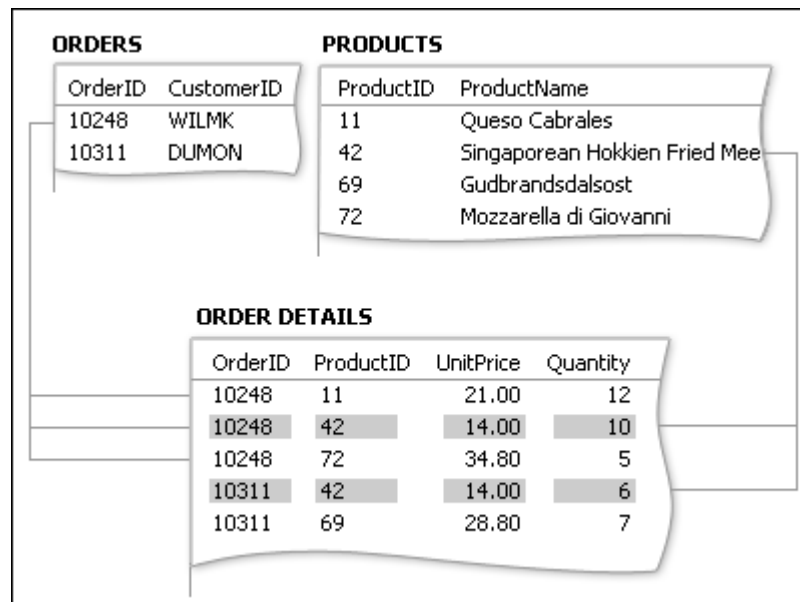
- You provide the basis for joining related tables by establishing pairings of primary keys and foreign keys.

## Creating a many-to-many relationship

- Consider the relationship between the Products table and Orders table.
- A single order can include more than one product. On the other hand, a single product can appear on many orders. Therefore, for each record in the Orders table, there can be many records in the Products table. And for each record in the Products table, there can be many records in the Orders table.
- This type of relationship is called a many-to-many relationship because for any product, there can be many orders; and for any order, there can be many products. Note that to detect many-to-many relationships between your tables, it is important that you consider both sides of the relationship.
- The answer is to create a **third table**, often called a **junction table**, that breaks down the **many-to-many** relationship into two one-to-many relationships. You insert the primary key from each of the two tables into the third table. As a result, the third table records each occurrence or instance of the relationship.

# Creating a many-to-many relationship

In the product sales database, the Orders table and the Products table are not related to each other directly. Instead, they are related indirectly through the Order Details table. The **many-to-many relationship** between orders and products is represented in the database by using two one-to-many relationships:



- The Orders table and Order Details table have a one-to-many relationship. Each order can have more than one line item, but each line item is connected to only one order.
- The Products table and Order Details table have a one-to-many relationship. Each product can have many line items associated with it, but each line item refers to only one product.

## Creating a many-to-many relationship

- From the Order Details table, you can determine all of the products on a particular order. You can also determine all of the orders for a particular product.
- After incorporating the Order Details table, the list of tables and fields might look something like this:





## Creating a one-to-one relationship

- Another type of relationship is the **one-to-one relationship**. For instance, suppose you need to record some special supplementary product information that you will need rarely or that only applies to a few products.
- Because you don't need the information often, and because storing the information in the Products table would result in empty space for every product to which it doesn't apply, you place it in a separate table.
- Like the Products table, you use the ProductID as the primary key. The relationship between this supplemental table and the Product table is a one-to-one relationship.
- For each record in the Product table, there exists a single matching record in the supplemental table. When you do identify such a relationship, both tables **must share a common field**.



## Creating a one-to-one relationship

- When you detect the need for a one-to-one relationship in your database, consider whether you can put the information from the two tables together in one table. If you don't want to do that for some reason, perhaps because it would result in a lot of empty space, the following list shows how you would represent the relationship in your design:
  - If the two tables have the same subject, you can probably set up the relationship by using the same primary key in both tables.
  - If the two tables have different subjects with different primary keys, choose one of the tables (either one) and insert its primary key in the other table as a foreign key.

*Determining the relationships between tables helps you ensure that you have the right tables and columns. When a **one-to-one** or **one-to-many** relationship exists, the tables involved **need to share a common column or columns**. When a **many-to-many** relationship exists, **a third table is needed** to represent the relationship.*

## Refining the design

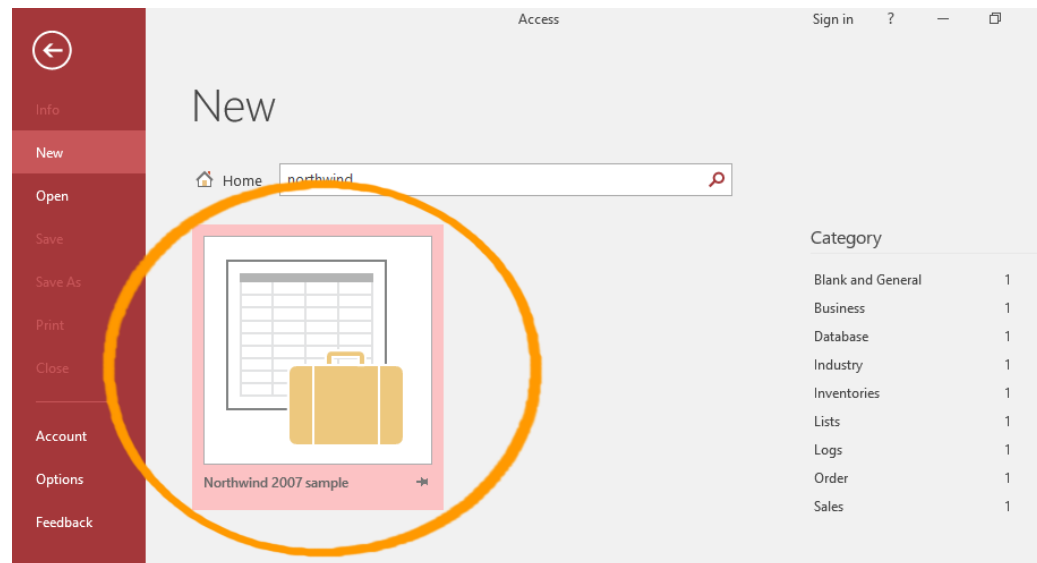
- Once you have the tables, fields, and relationships you need, you should create and populate your tables with sample data and try working with the information: creating queries, adding new records, and so on.
- Doing this helps highlight potential problems — for example, you might need to add a column that you forgot to insert during your design phase, or you may have a table that you should split into two tables to remove duplication.
- See if you can use the database to get the answers you want. Create rough drafts of your forms and reports and see if they show the data you expect. Look for unnecessary duplication of data and, when you find any, alter your design to eliminate it.

## Applying the normalization rules

- You can apply the data normalization rules (sometimes just called normalization rules) as the next step in your design. You use these rules to see if your tables are structured correctly. The process of applying the rules to your database design is called normalizing the database, or just normalization.
- Normalization is most useful after you have represented all of the information items and have arrived at a preliminary design. The idea is to help you ensure that you have divided your information items into the appropriate tables. What normalization cannot do is ensure that you have all the correct data items to begin with.
- You apply the rules in succession, at each step ensuring that your design arrives at one of what is known as the "normal forms." Five normal forms are widely accepted — the first normal form through the fifth normal form. This article expands on the first three, because they are all that is required for the majority of database designs.

# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



**Review and test everything from this practical lesson with this database. Please, discover different types of relationships between tables.**



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 5. Practice

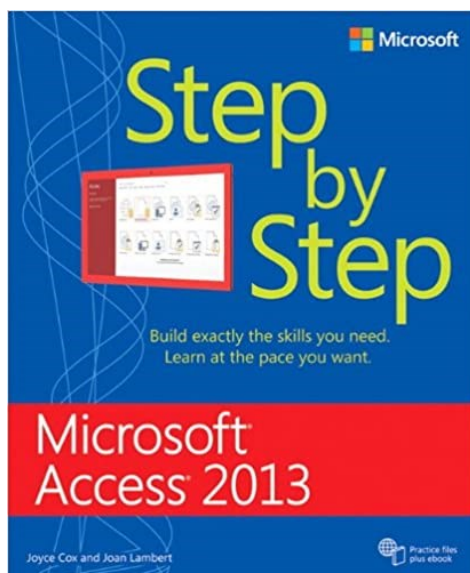
### ❑ Topic 2. Maintaining database changes. Ensuring the reliability of information in the database.

#### ❑ Practical lesson 2. Ensuring the reliability of information in the database

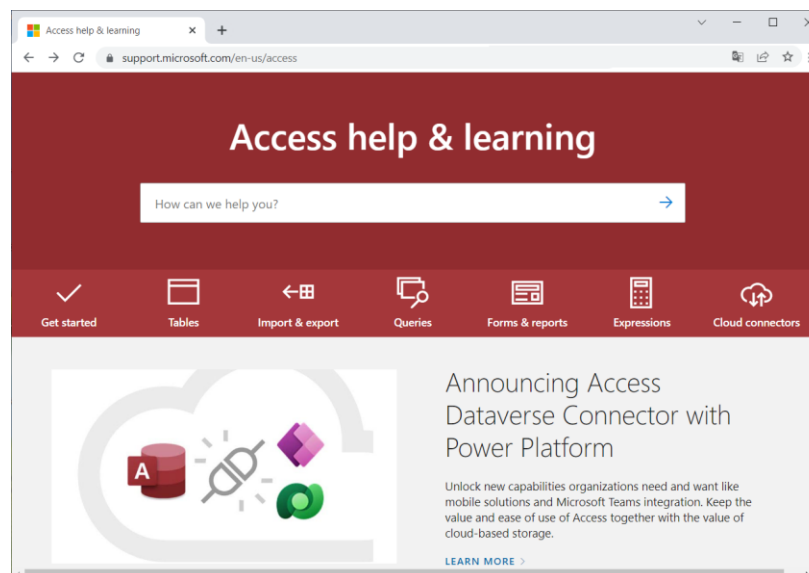


ERASMUS+

# Practical lesson: Ensuring the reliability of information in the database.



Joyce Cox, Joan Lambert.  
*Microsoft Access 2013 Step By Step*, Microsoft Press., 2013.



<https://support.microsoft.com/en-us/access>



## Introduction

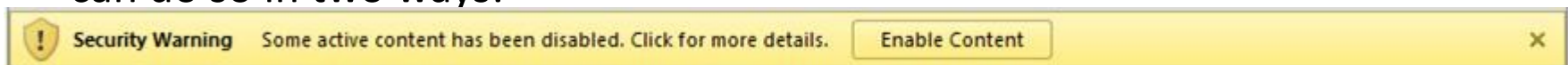
- Data is arguably the most important asset that one organization has.
- The database is a key part of every organization's infrastructure, and if it goes down, it can create major problems.
- Data is key for handling daily activities and for making short and long-term decisions.
- For an organization to run effectively and efficiently, it needs to have a **reliable database**.
- Database **reliability** is defined broadly to mean that the database performs consistently without causing problems.

## Introduction

- For data to be considered reliable, there must be:
  - Data **integrity**, which means that all data in the database is accurate and that there is consistency throughout data. Data consistency is defined broadly to include the type and amount of data.
  - Data **safety**, which means that only authorized individuals access the database. Data security also includes preventing any type of data corruption and ensuring that data is always accessible.
  - Data **recoverability**, which means there are effective procedures in place to recover any lost data. This is a key to database reliability, ensuring that even if other safety measures fail, there is a system for recovering data.

## Decide whether to trust a database

- By default, Access disables all the potentially **unsafe** code or other components in a database, regardless of the version of Access that you used to create the database.
- When Access **disables** content, it informs you of the action by displaying the Message Bar.
- If you see the Message Bar, you can choose whether to trust the disabled content in the database. If you decide to trust the disabled content, you can do so in two ways:



- **Use the Message Bar:** Click Enable Content on the Message Bar. When you choose this option, you may need to repeat the procedure if the database changes.
- **Trust the database permanently:** Place the database in a trusted location — a folder on a drive or network that you mark as trusted.

# Factors to consider when deciding whether to trust a database

- Before you decide whether to trust a database, you should consider the following factors:
  - **Your own security policy:** You or your company may have a security policy in place that specifies how to handle Access database files. For example, you might have a very robust backup system in place, and decide that you are willing to trust most database files, unless you have a specific reason not to. Conversely, you may not have a good backup system, and therefore might want to be very cautious when you decide whether to trust a database.
  - **Your goal:** When Access disables content in a database that you have not trusted, it does not block your access to the data in that database. If you want to review the data in a database and do not want to perform any actions that might be unsafe, such as running an action query or using certain macro actions, you do not have to trust the database. If you are not sure whether an action is considered unsafe, you can try to perform the action while the database content has been blocked by disabled mode. If the action is potentially unsafe, it will be blocked in this circumstance.

# Factors to consider when deciding whether to trust a database

- Before you decide whether to trust a database, you should consider the following factors:
  - **The database source:** If you created the database, or if you know that it came from a source that you trust, you can decide to trust the database. If the database came from a possibly unreliable source, you might want to leave the database untrusted until you make sure that its content is safe.
  - **The contents of the database file:** If you cannot make a trust decision based on other information, you might consider thoroughly examining the database contents to see what potentially unsafe content the database might contain. After you conduct a complete check and are sure that the content is safe, you can decide to trust the database.
  - **The security of the location where the database is stored:** Even if you know that the contents of a database file are safe, if the file is stored in a location that is not fully secure, someone might introduce unsafe content into the database. You should be careful when deciding to trust database files that are stored in locations that might not be secure.

## Ways to share an Access desktop database

- There are several ways that you can share an Access database depending on your needs and resource **availability**.
- Database applications change and grow over time. Many factors impact needs and performance including the number of concurrent users, the network environment, throughput, latency, the size of the database, peak usage times, and expected growth rates. In short, if your database solution is successful, it probably needs to evolve. Fortunately, Access has an evolutionary path, from simple to advanced, that you can take over time to effectively scale your solution. The following table summarizes Access scenarios and workloads to help you choose that path.

Sharing method	Front-end	Back end	Data location	Scenario	Users	Workload
Single database	---	---	Home network	Home/small biz	A few	Light
SharePoint	Access	SharePoint Lists	SharePoint site	Enterprise team	Dozens	Light
Split database	Access	Access	Network folder	Enterprise team	Dozens	Medium
Client/Server	Access	SQL Server	Database Server	Enterprise-wide	Many	Heavy
Hybrid Client/Server	Access	Azure SQL	Azure Cloud	Enterprise-wide	Many	Heavy

ERASMUS+

# Ways to share an Access desktop database

## Share a single database

- This is the simplest option and has the least requirements, but also provides the least functionality.
- In this method, the database file is stored on a shared network drive, and all users share the database file simultaneously.
- Some limitations include **reliability** and **availability** if there are multiple simultaneous users changing data since all database objects are shared.
- This technique can also reduce performance as all the database objects are sent across the network.
- This option might work for you if only a few people are expected to use the database at the same time and users don't need to customize the design of the database. But this method is less secure than other methods of sharing a database, because each user has a full copy of the database file, increasing the risk of unauthorized access.



# Ways to share an Access desktop database

## Share a single database

- Make sure that Access is set to open in shared mode on all of the users' computers. This is the default setting, but you should check to be sure — if a user opens the database in exclusive mode, it will interfere with data availability.
  - a. Start Access and under **File**, click **Options**.
  - b. In the **Access Options** box, click **Client Settings**.
  - c. In the **Advanced** section, under **Default open mode**, select **Shared**, click **OK**, and then exit Access.
- Copy the database file to the shared folder. After you copy the file, make sure that the file attributes are set to allow read/write access to the database file. Users must have read/write access to use the database.
- On each user's computer, create a shortcut to the database file.

# Ways to share an Access desktop database

## Share a split database

- This is a good choice if you do not have a SharePoint site or a database server.
- You can share a split database over a Local Area Network (LAN).
- When you split a database, you reorganize it into two files — a back-end database that contains the data tables, and a front-end database that contains all the other database objects such as queries, forms, and reports.
- Each user interacts with the data by using a local copy of the front-end database.
- The benefits of splitting a database include the following:
  - **Improved performance:** Only the data is shared across the network not the tables, queries, forms, reports, macros and modules.

# Ways to share an Access desktop database

## Share a split database

- The benefits of splitting a database include the following:
  - **Greater availability:** Database transactions such as record edits are completed more quickly.
  - **Enhanced security:** Users access the back-end database through linked tables; it is less likely that intruders can obtain unauthorized access to the data via the front-end database.
  - **Improved reliability:** If a user encounters a problem and the database closes unexpectedly, any database file corruption is usually limited to the copy of the front-end database that the user had open.
  - **Flexible development environment:** Each user can independently develop queries, forms, reports, and other database objects without affecting other users. You can also develop and distribute a new version of the front-end database without disrupting access to the data that is stored in the back-end database.

# Ways to share an Access desktop database

## Share data by using a database server

- You can use Access with a database server product such as SQL Server to share your database. This method offers you many benefits, but does require additional software — a database server product.
- This method is similar to splitting a database because the tables are stored on the network, and each user has a local copy of an Access database file that contains links to the tables, along with queries, forms, reports, and other database objects.
- Benefits of this sharing method depends on the database server software that you use, but generally include user accounts and selective access to data, **excellent data availability**, and good integrated data management tools.
- Moreover, most database server software works well with earlier versions of Access, so not all your users must use the same version.

# Ways to share an Access desktop database

## Share data by using a database server

- Benefits of sharing a database by using a database server:
  - **High performance and scalability:** In many situations, a database server offers better performance than an Access database file alone. Many database server products also provide support for very large, terabyte-sized databases, approximately 500 times the current limit for an Access database file (two gigabytes). Database server products generally work very efficiently by processing queries in parallel and minimizing additional memory requirements when more users are added.
  - **Increased availability:** Most database server products allow you to back up your database while it is in use. Consequently, you do not have to force users to exit the database to back up data. Moreover, database server products usually handle concurrent editing and record-locking very efficiently.
  - **Improved security:** No database can be made completely secure. However, database server products offer robust security that will help protect your data from unauthorized use. Most database server products offer account-based security, allowing you to specify who can see which tables. Even in the event that the Access front-end is improperly obtained, unauthorized use of data is prevented by account-based security.

# Ways to share an Access desktop database

## Share data by using a database server

- Benefits of sharing a database by using a database server:
  - **Automatic recoverability:** In case of system failure (such as an operating system crash or power outage), some database server products have automatic recovery mechanisms that recover a database to the last state of consistency in a matter of minutes, with no database administrator intervention.
  - **Server-based processing:** Using Access in a client/server configuration helps reduce network traffic by processing database queries on the server before sending results to the client. Having the server do the processing is usually more efficient, especially when working with large data sets.
  - **Azure SQL Server:** In addition to the benefits of SQL Server, offers dynamic scalability with no downtime, intelligent optimization, global scalability and availability, elimination of hardware costs, and reduced administration.

## Protect databases

- Database protection takes two forms: **ensuring** that the database's data is **secure**, and **ensuring** that its data is **available** and **useable**.
- The need for **database security** is an unfortunate fact of life. As with your house, car, office, or briefcase, the level of security required for your database depends on the value of what you have and whether you are trying to protect it from curious eyes, accidental damage, malicious destruction, or theft.
- The **security** of a company's business information can be critical to its survival. For example, you might not be too concerned if a person gained unauthorized access to your products list, but you would be very concerned if a competitor managed to obtain your customer list. And the destruction or deletion of your critical order information would be a disaster.
- Your goal is to provide **adequate protection** without imposing unnecessary restrictions on the people who need access to your database. In addition to ensuring that a database is secure, you need to ensure that it is well maintained.



## Assigning passwords to databases

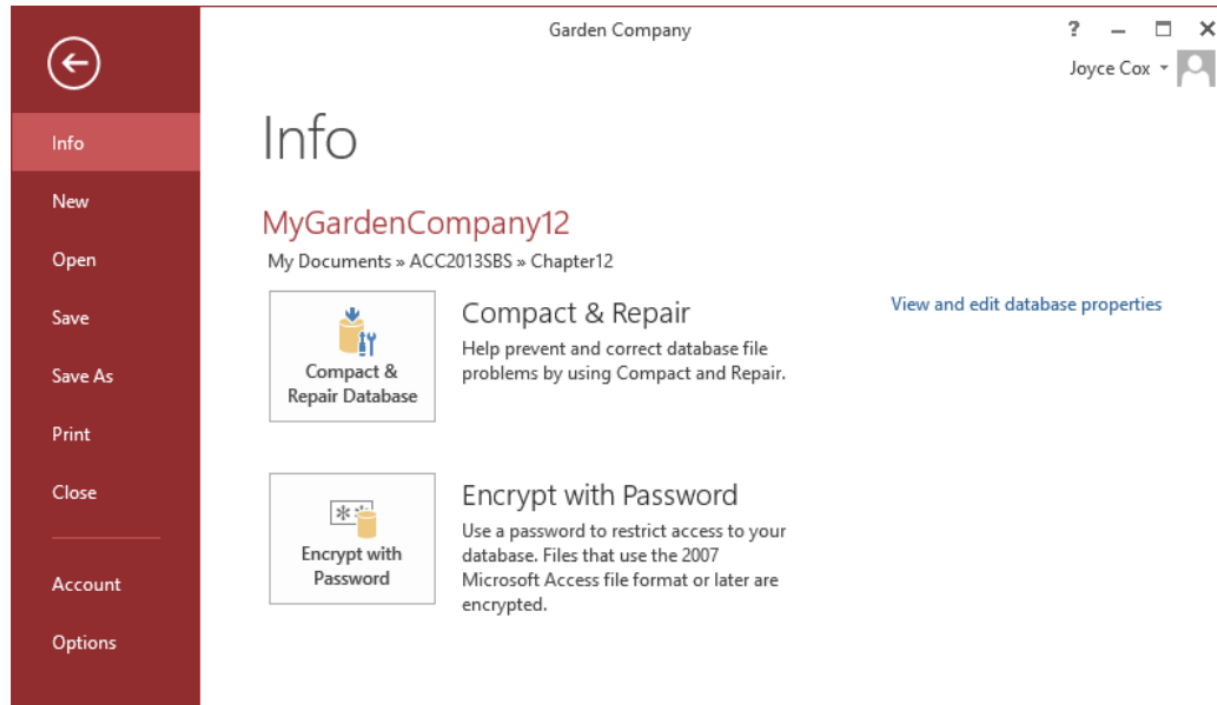
- You can keep **unauthorized** users out of a database by **assigning** it a **password**. Access then prompts anyone attempting to open the database for the password, and opens it only if the **password** is correct.
- To assign a **password** to or remove a password from a database, you must first open the database for exclusive use, meaning that no one else can have the database open. This will not be a problem for a database stored on your own computer and used only by you, but if you want to set or remove a password for a database that is located on a network, you will first need to make sure nobody else is using it.
- You can use any word or phrase as a database password, but to create a secure password, keep the following in mind:
  - Passwords are case sensitive
  - You can include letters, accented characters, numbers, spaces, and most punctuation marks.
  - A good password includes uppercase letters, lowercase letters, and symbols or numbers, and isn't a word found in a dictionary.

## Assigning passwords to databases

- Assigning a password to a database has an important secondary benefit. A database created in Access is a binary file (a file that stores instructions and data in such a way that it can usually be understood only by a computer).
- If you open the file in a word processor or a text editor, its content is mostly unreadable, but if you look closely enough at the file, you can discover quite a bit of information.
- It is unlikely that enough information will be exposed to allow anyone to steal anything valuable. However, people can and do scan files with computer tools designed to look for key words that lead them to restricted information.
- When you assign a password to a database, the database is automatically encrypted each time it is closed, making it more unreadable.
- Opening the file in Access with the correct password decrypts the file and makes its data readable again.

# Assigning passwords to databases

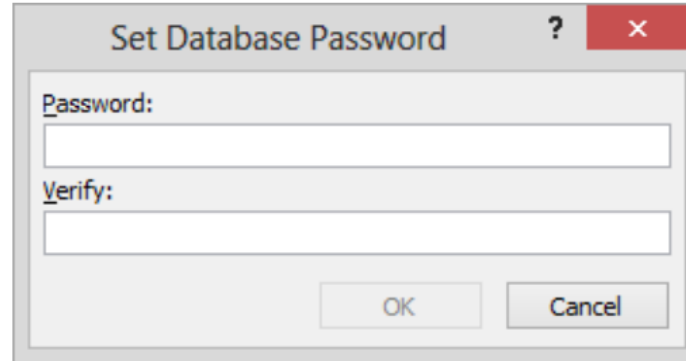
- With Access running but no database open, display the **Open** page of the **Backstage** view.
- Select your database and then click the **Open** arrow, and in the list, click **Open Exclusive**.



*From this page, you can run utilities to help prevent database problems, assign a password, and assign file properties that help identify the file.*

## Assigning passwords to databases

- Click Encrypt with Password to open the **Set Database Password** dialog box.

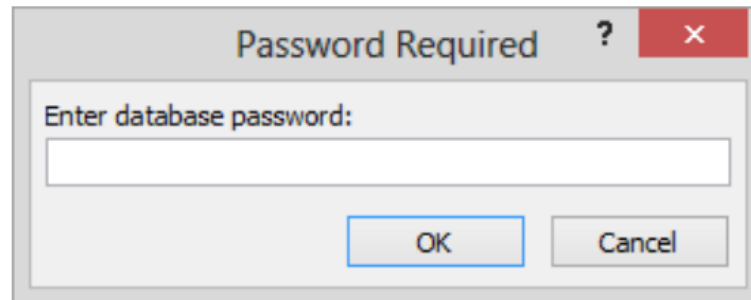
A screenshot of the 'Set Database Password' dialog box. It has a title bar with a question mark and a close button. Inside, there are two text input fields: the first is labeled 'Password:' and the second is labeled 'Verify:'. Below the fields are 'OK' and 'Cancel' buttons.

*As you enter the characters of the password in this dialog box,  
Access disguises them as asterisks.*

- In the **Password** box, your *password*, and then press the **Tab** key.
- In the **Verify** box, enter your *password* again! Then click **OK**. A message box warns that row-level locking will be ignored.
- Click **OK** to close the message box, and then close the database without exiting
- Access.

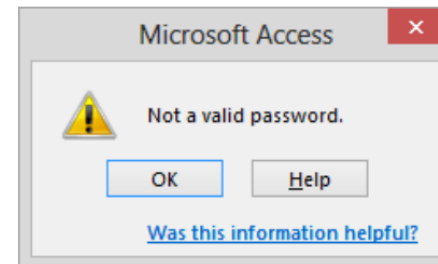
## Assigning passwords to databases

- Now if you try to open the database. Instead of displaying the Home Page navigation form, Access opens the Password Required dialog box.



*You cannot work with the database unless you know the password.*

- In the **Enter database password** box, enter your *password*, and click **OK**.
- If you enter incorrect password Access show:



*Access warns that the password is not valid.*

## Preventing database problems

- Normal database use can cause the internal structure of any database in any database program to become fragmented, sometimes resulting in a bloated file and inefficient use of disk space.
- Fortunately, Access monitors the condition of database files as you open and work with them, but you still need to pay attention, particularly if the performance of the database seems slow or erratic.
- You can take a variety of steps to help keep an Access database healthy and running smoothly. Your first line of defense against damage or corruption in any kind of file is to back it up.
- Database files can rapidly become very large, so you need to choose an appropriate place to store a backup copy, such as a DVD, another computer on your network, or removable media such as a USB flash drive or external hard disk.

## Preventing database problems

- In addition to regularly backing up the database, you can use the following Access utilities to keep it running smoothly:
  - **Compact and Repair Database:** Optimizes performance by rearranging how the file is stored on your hard disk, and then attempts to repair any corruption in tables, forms, and reports.
  - **Database Documenter:** Produces a detailed report that contains enough information to rebuild the database structure if necessary.
  - **Analyze Performance:** Analyzes the objects in the database and offers three types of feedback: ideas, suggestions, and recommendations. You can instruct Access to optimize the file by following through on any of the suggestions or recommendations.
  - **Analyze Table:** Tests database tables for compliance with standard database design principles, suggests solutions to problems, and implements those solutions at your request.

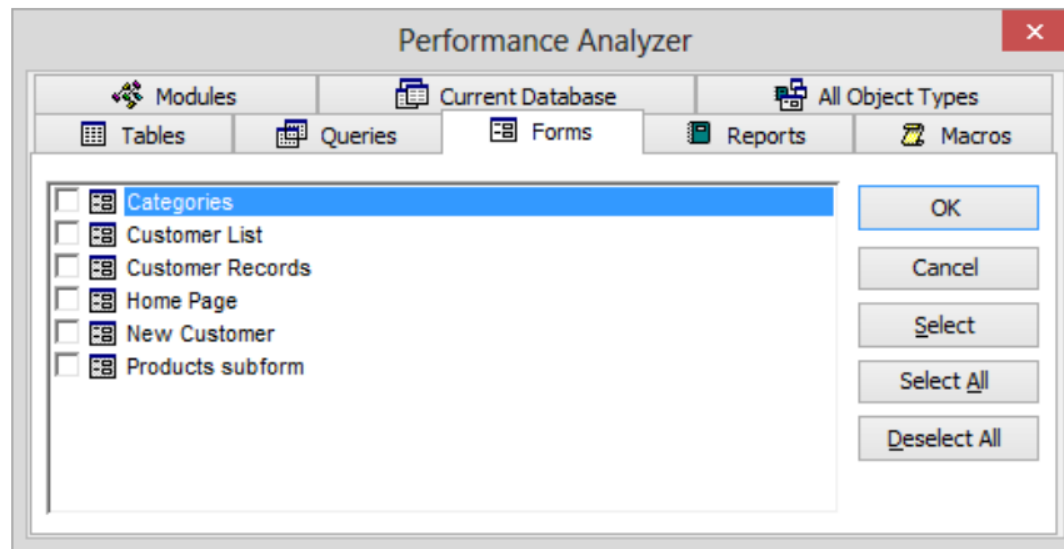


## Preventing database problems

- Open the database file, and then follow the steps:
  1. Close any open database objects, and then display the **Save As** page of the **Backstage view**.
  2. In the **Advanced** area of the right pane, double-click **Back Up Database**.
  3. In the **Save As** dialog box click **Save**, which creates a copy of the database with the current date appended to the file name in the specified folder.
  4. Display the **Info** page of the **Backstage** view, and then click **Compact & Repair Database**.
  5. Close any open database objects, and then in the **Forms** area of the **Navigation pane**, click **Home Page**.
  6. On the **Database Tools** tab, in the **Analyze** group, click the **Analyze Performance** button to open the **Performance Analyzer** dialog box. Notice that each type of database object is represented by a page, and there are also pages for all objects and for the database as a whole.

# Preventing database problems

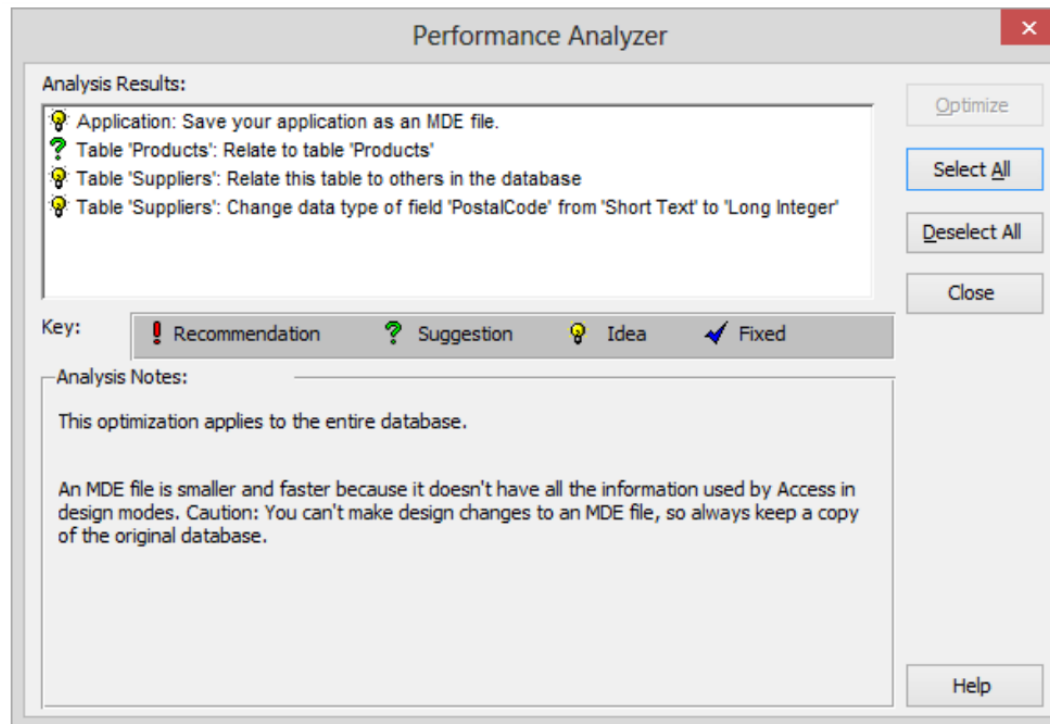
- Open the database file, and then follow the steps:
  6. On the **Database Tools** tab, in the **Analyze** group, click the **Analyze Performance** button to open the **Performance Analyzer** dialog box. Notice that each type of database object is represented by a page, and there are also pages for all objects and for the database as a whole.



*The active page reflects the object that is selected in the Navigation pane when you start the utility.*

# Preventing database problems

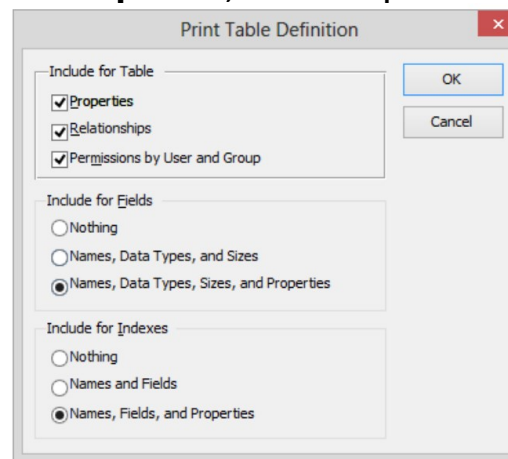
- Open the database file, and then follow the steps:
  7. Click the **All Object Types** tab, click **Select All**, and then with the check boxes for all the objects in the database selected, click **OK** to start the analyzer.



*The Key tells you the nature of each item in the Analysis Results list.*

# Preventing database problems

- Open the database file, and then follow the steps:
  8. Click each entry in turn, and read the information in the **Analysis Notes** area.
  9. Close the **Performance Analyzer** dialog box. Finally, let's create a report of the structure of the database.
  10. On the **Database Tools** tab, in the **Analyze** group, click the **Database Documenter** button to open the **Documenter** dialog box. Notice that this dialog box is identical to the **Performance Analyzer** dialog box, with a page for each type of object the utility can document and a page for all the existing database objects.
  11. Click the **Tables** tab, and then click **Options**, which opens the **Print Table Definition** dialog box.

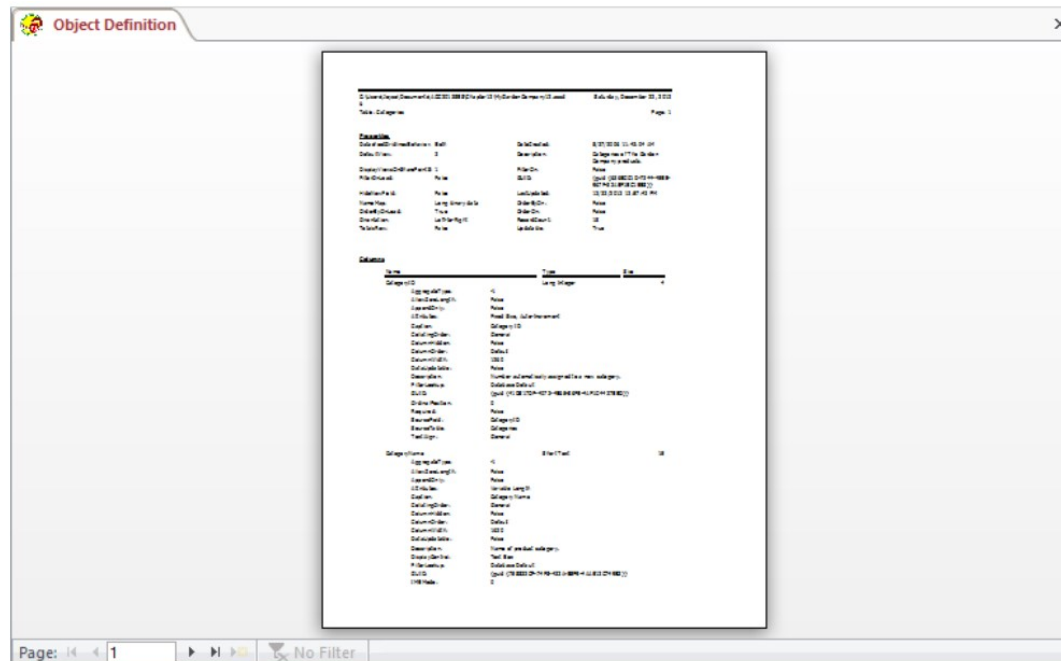


*These print options are associated with tables.*

ERASMUS+

# Preventing database problems

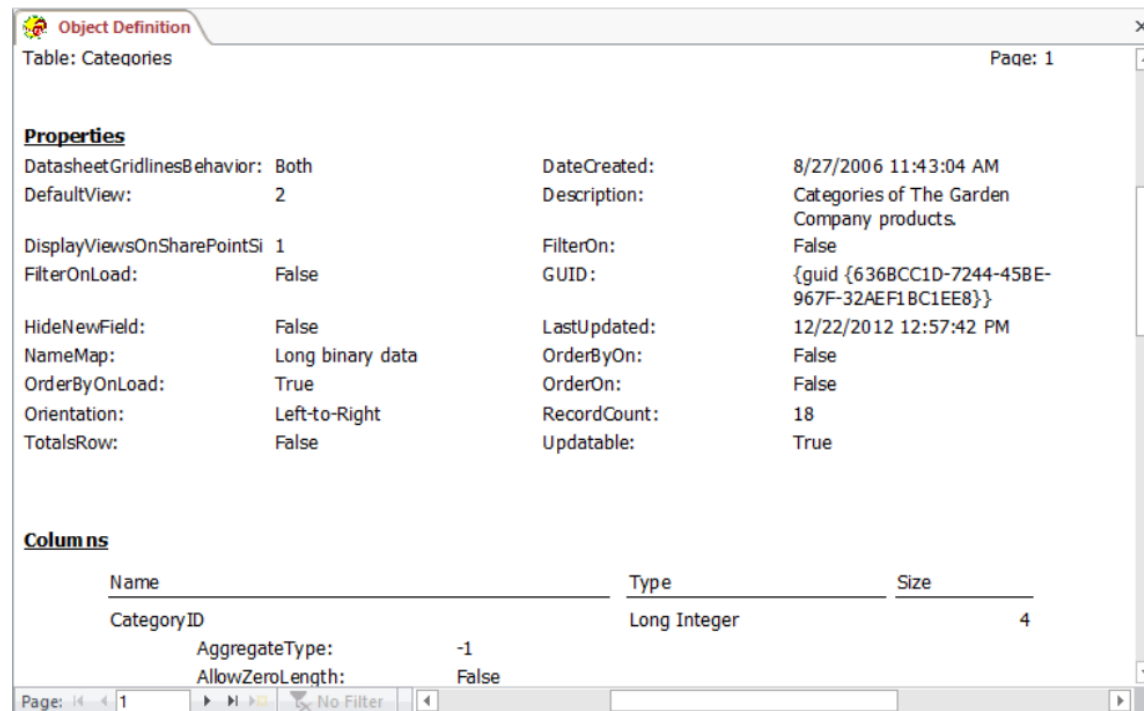
- Open the database file, and then follow the steps:
  - In the **Print Table Definition** dialog box, click **Cancel**.
  - Click the **All Object Types** tab, click **Select All**, and then click **OK** to start the documentation process and create the report, which Access displays in **Print Preview**.



*The report for this simple database is more than 200 pages long.*

# Preventing database problems

- Open the database file, and then follow the steps:
  - Zoom in on the report to examine the kinds of things included in the documentation. Then use the page navigation bar to scroll through a few pages.



*The report details the structure of every object in the database.*

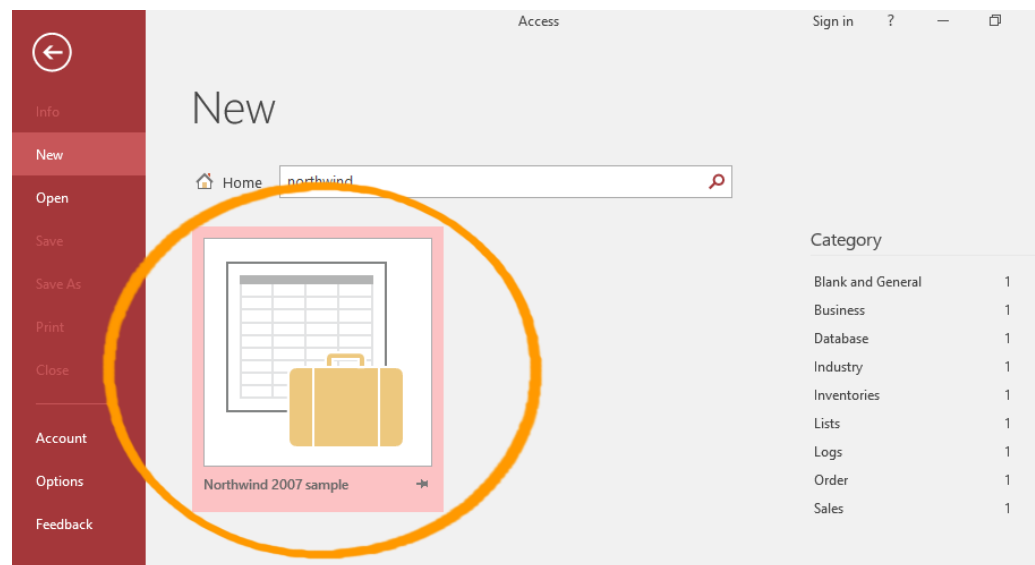
## Key points

- You can **assign a password** to a database to prevent unauthorized users from opening it. Assigning a password automatically encrypts the database.
- **Splitting** a database can enhance database performance and safeguard data in a multiuser environment.
- If you save the database as an **.accde** file, people can use its forms and reports but not create new ones.
- Access **automatically fixes** many problems that can arise with a database. You can prevent problems by frequently using the utilities provided for that purpose.
- The simplest way to protect your database is to **back it up regularly**.



# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



Review and test everything from this practical lesson with this database.



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

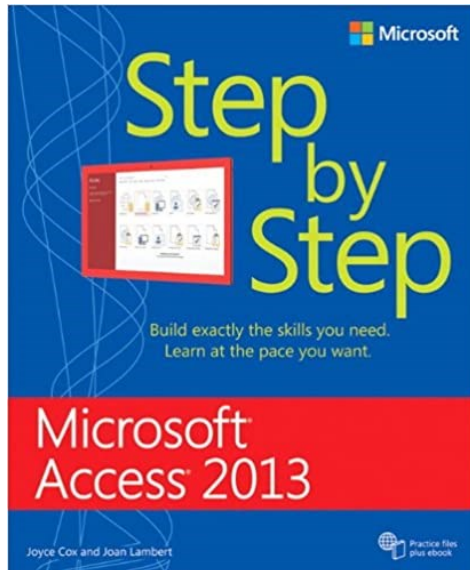
## ❑ Module 5. Practice

### ❑ Topic 3. Creation of a query. Use of queries.

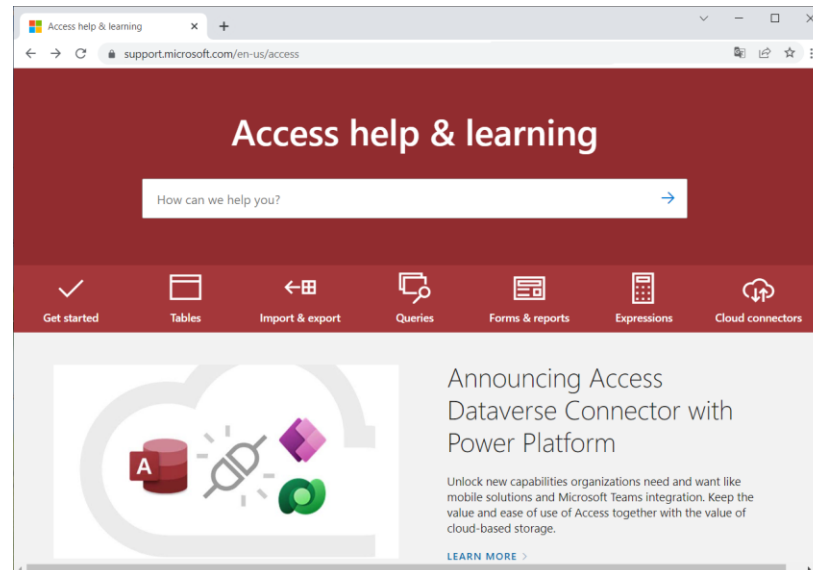
#### ❑ Practical lesson 1. Creation of a query



# Practical lesson. Creation of a query.



*Joyce Cox, Joan Lambert.  
Microsoft Access 2013 Step By  
Step, Microsoft Press., 2013.*



*<https://support.microsoft.com/en-us/access>  
<https://www.customguide.com>*

## Introduction to queries

- Using a query makes it easier to view, add, delete, or change data in your Access database. Some other reasons for using queries:
  - Find specific quickly data by filtering on specific criteria (conditions)
  - Calculate or summarize data
  - Automate data management tasks, such as reviewing the most current data on a recurring basis.

## Introduction to queries

- In a well-designed database, the data that you want to present through a form or report is usually located in multiple tables.
- A query can pull the information from various tables and assemble it for display in the form or report.
- A query can either be a request for data results from your database or for action on the data, or for both.
- A query can give you an answer to a simple question, perform calculations, combine data from different tables, add, change, or delete data from a database.
- Since queries are so versatile, there are many types of queries and you would create a type of query based on the task.

# Introduction to queries

Major query types	Use
Select	To retrieve data from a table or make calculations.
Action	Add, change, or delete data. Each task has a specific type of action query. Action queries are not available in Access web apps.



## Create a select query

- If you want to review data from only certain fields in a table, or review data from multiple tables simultaneously or maybe just see the data based on certain criteria, a select query type would be your choice.
- For **example**, if your database has a table with a lot of information about products and you want to review a list of products and their prices, here's how you'd create a select query to return just the product names and the respective price:
  1. Open the database and on the **Create** tab, click **Query Design**.
  2. On the **Tables** tab, double-click the **Products** table.
  3. In the Products table, let's say that you have Product Name and List Price fields. Double-click the **Product Name** and **List Price** to add these fields to the query design grid.
  4. On the **Design** tab, click **Run**. The query runs, and displays a list of products and their prices.

## Create a select query

- You can also review data from multiple related tables simultaneously. For **example**, if you have a database for a store that sells food items and you want to review orders for customers who live in a particular city. Say that the data about orders and data about customers are stored in two tables named Customers and Orders respectively. If each table has a Customer ID field, which forms the basis of a one-to-many relationship between the two tables. You can create a query that returns orders for customers in a particular city, for example, Las Vegas, by using the following procedure:
  1. Open the database. On the **Create** tab, in the **Query** group, click **Query Design**.
  2. On the **Tables** tab, double-click **Customers** and **Orders**.
  3. In the Customers table, double-click **Company** and **City** to add these fields to the query design grid.
  4. In the query design grid, in the **City** column, clear the check box in the **Show** row.

## Create a select query

5. Open the database. On the **Create** tab, in the **Query** group, click **Query Design**.

*Clearing the **Show** check box prevents the query from displaying the city in its results, and typing **Las Vegas** in the **Criteria** row specifies that you want to see only records where the value of the City field is Las Vegas. In this case, the query returns only the customers that are located in Las Vegas. You don't need to display a field to use it with a criterion.*

6. In the Orders table, double-click **Order ID** and **Order Date** to add these fields to the next two columns of the query design grid.
7. On the **Design** tab, in the **Results** group, click **Run**. The query runs, and then displays a list of orders for customers in Las Vegas.
8. Press CTRL+S to save the query.

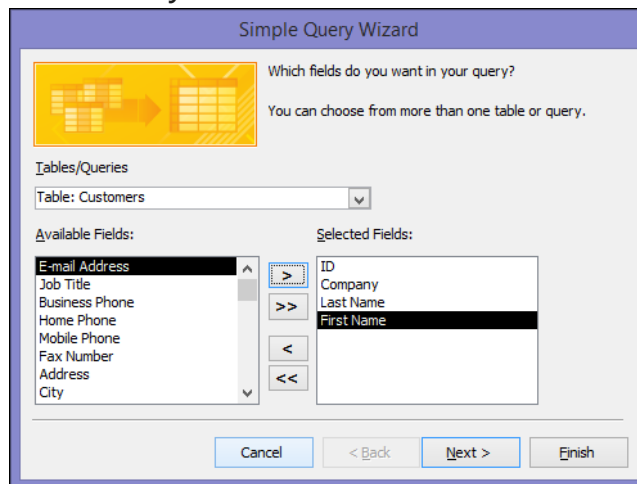
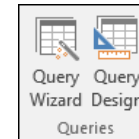
**Now, we will show how to create a simple select query using the Query Wizard in Access.**

## Use the Query Wizard to create a select query

- You can use the Query Wizard in Access to automatically create a select query.
- When you use the wizard, you have less control over the details of the query design, but the query is usually created faster than if you did not use the wizard.
- Moreover, the wizard can catch some simple design mistakes and prompt you to perform a different action.
- If you use fields from data sources that are not related to each other, the Query Wizard asks you if you want to create relationships. Therefore, before you run the wizard, consider creating any relationships that your query needs.

# Use the Query Wizard to create a select query

1. On the **Create** tab, in the **Queries** group, click **Query Wizard**.
2. In the **New Query** dialog box, click **Simple Query Wizard**, and then click **OK**.
3. Next, you add fields. You can add up to 255 fields from as many as 32 tables or queries. For each field, perform these two steps:
  1. Under **Tables/Queries**, click the table or query that contains the field.
  2. Under **Available Fields**, double-click the field to add it to the **Selected Fields** list. If you want to add all fields to your query, click the button with the double right arrows (>>).
  3. When you have added all the fields that you want, click **Next**.

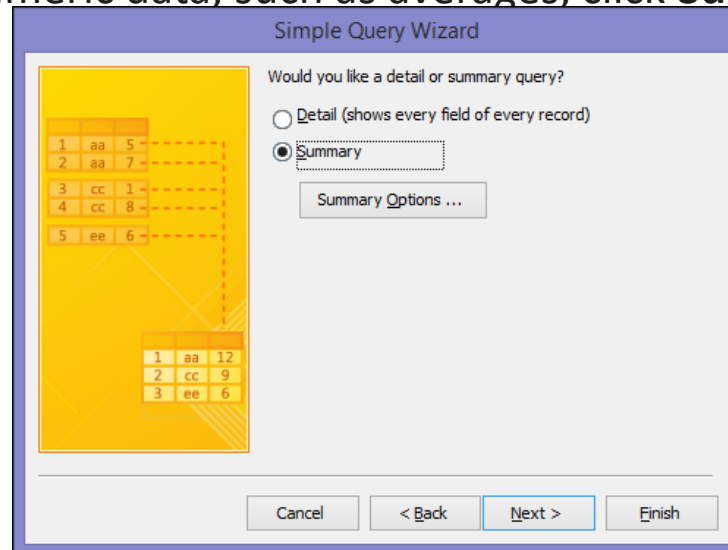


# Use the Query Wizard to create a select query

4. If you did not add any number fields (fields that contain numeric data), skip ahead to step 9. If you added any number fields, the wizard asks whether you want the query to return details or summary data.

Do one of the following:

1. If you want to see individual records, click **Detail**, and then click **Next**. Skip ahead to step 9.
2. If you want to see summarized numeric data, such as averages, click **Summary**, and then click **Summary Options**.



The image shows a screenshot of the 'Simple Query Wizard' dialog box. On the left, there is a preview of a data table with 5 rows and 3 columns. The first two columns contain text ('aa', 'cc', 'ee') and the third column contains numbers (5, 7, 1, 8, 6). A smaller table is shown below it with 3 rows and 3 columns. On the right, the wizard asks 'Would you like a detail or summary query?'. The 'Detail' option is unselected, and the 'Summary' option is selected. Below the 'Summary' option is a button labeled 'Summary Options ...'. At the bottom of the dialog are four buttons: 'Cancel', '< Back', 'Next >', and 'Finish'.

1	aa	5
2	aa	7
3	cc	1
4	cc	8
5	ee	6

1	aa	12
2	cc	9
3	ee	6

Simple Query Wizard

Would you like a detail or summary query?

☐ Detail (shows every field of every record)

☒ Summary

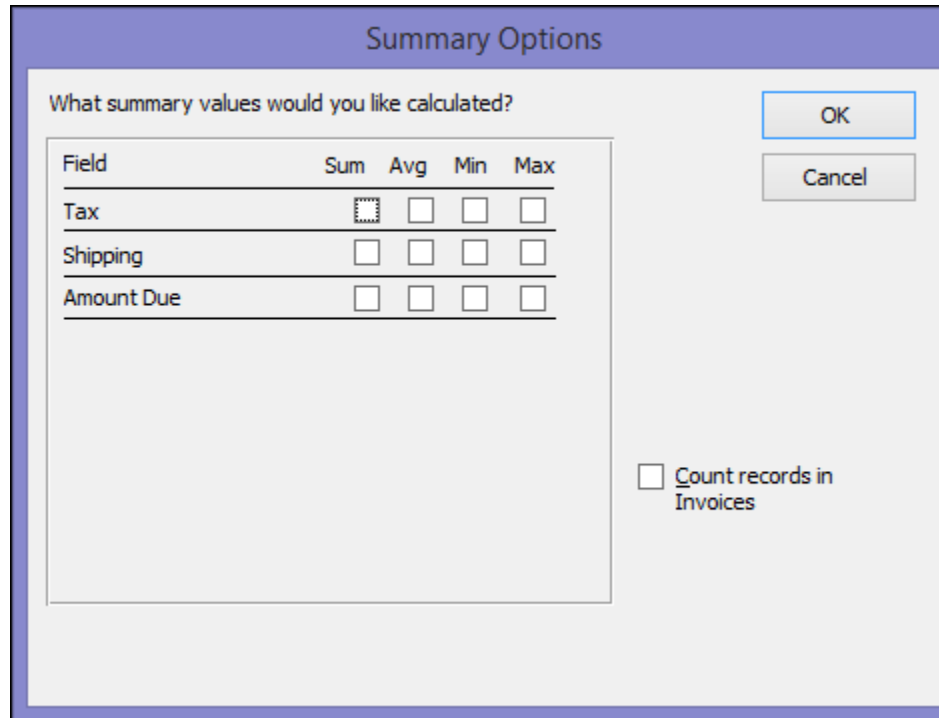
Summary Options ...

Cancel < Back Next > Finish

# Use the Query Wizard to create a select query

5. In the **Summary Options** dialog box, specify which fields you want to summarize, and how you want to summarize the data. Only number fields are listed.

For each number field, choose one of the following functions:

A screenshot of the 'Summary Options' dialog box. The title bar is purple and says 'Summary Options'. Inside, the text 'What summary values would you like calculated?' is followed by a table. The table has five columns: 'Field', 'Sum', 'Avg', 'Min', and 'Max'. There are three rows of data: 'Tax', 'Shipping', and 'Amount Due'. Each row has a checkbox in the 'Sum' column and empty checkboxes in the 'Avg', 'Min', and 'Max' columns. To the right of the table are 'OK' and 'Cancel' buttons. Below the table is a checkbox labeled 'Count records in Invoices'.

Field	Sum	Avg	Min	Max
Tax	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Shipping	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Amount Due	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

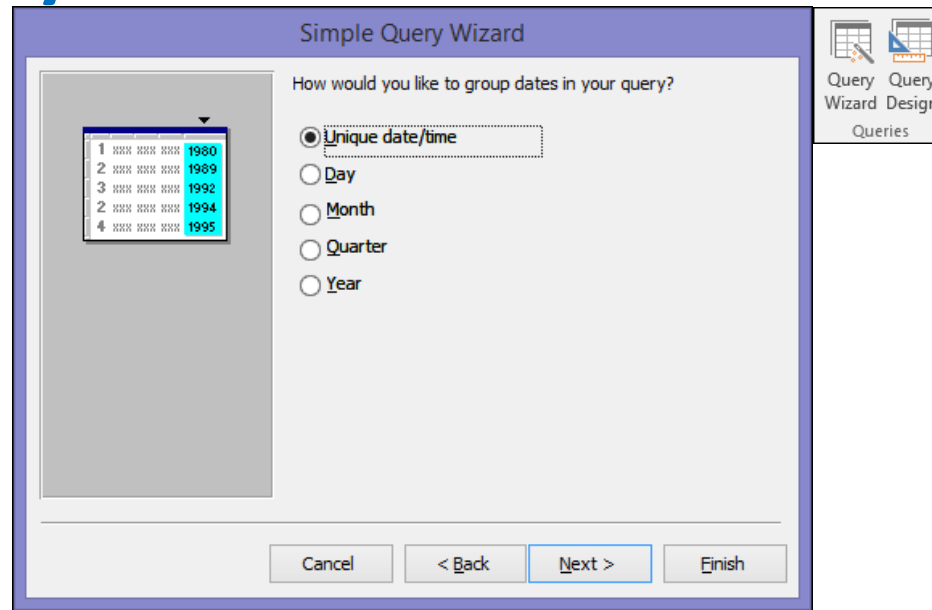
☐ Count records in Invoices



## Use the Query Wizard to create a select query

6. In If you want the query results to include a count of the records in a data source, select the appropriate **Count records in data source name** check box.
7. Click **OK** to close the **Summary Options** dialog box.
8. If you did not add a date/time field to the query, skip ahead to step 9. If you added a date-time field to the query, the Query Wizard asks you how you would like to group the date values. For example, suppose you added a number field ("Price") and a date/time field ("Transaction\_Time") to your query, and then specified in the **Summary Options** dialog box that you want to see the average value of the number field "Price". Because you included a date/time field, you could calculate summary values for each unique date/time value, for each day, for each month, for each quarter, or for each year.

# Use the Query Wizard to create a select query



Select the time period that you want to use to group the date/time values, and then click **Next**.

9. On the last page of the wizard, give the query a title, specify whether you want to open or modify the query, and then click **Finish**.

If you choose to open the query, the query displays the selected data in Datasheet view. If you choose to modify the query, the query opens in Design view.

## Create an update query

- You can use an update query to change the data in your tables, and you can use an update query to enter criteria to specify which rows should be updated. An update query provides you an opportunity to review the updated data before you perform the update.
- For example in the Chicago Orders table, the Product ID field shows the numeric Product ID. To make the data more useful in reports, you can replace the product IDs with product names, use the following procedure:
  1. Open the Chicago Orders table in Design view.
  2. In the Product ID row, change the Data Type from **Number** to **Text**.
  3. Save and close the Chicago Orders table.
  4. On the **Create** tab, in the **Query** group, click **Query Design**.
  5. Double-click **Chicago Orders** and **Products**.
  6. On the **Design** tab, in the **Query Type** group, click **Update**.

## Create an update query

7. In the design grid, the **Sort** and **Show** rows disappear, and the **Update To** row appears.
8. In the **Chicago Orders** table, double-click **Product ID** to add this field to the design grid.
9. In the design grid, in the **Update To** row of the **Product ID** column, type or paste the following: **[Products].[Product Name]**
10. In the **Criteria** row, type or paste the following: **[Product ID] Like ([Products].[ID])**
11. You can review which values will be changed by an update query by viewing the query in Datasheet view.
12. On the **Design** tab, click **View > Datasheet View**. The query returns a list of Product IDs that will be updated.
13. On the **Design** tab, click **Run**.

*When you open the Chicago Orders table, you will see that the numeric values in the Product ID field have been replaced by the product names from the Products table.*

## Create an update query

- An Update query is a type of action query that makes changes to several records at the same time. For example, you could create an Update query to raise prices on all the products in a table by 10%.
- Just like other action queries, you create an Update query by first creating a Select query and then converting the Select query to an Update query.
- UPDATE is especially useful when you want to change many records or when the records that you want to change are in multiple tables.
- You can change several fields at the same time. The following example increases the Order Amount values by 10 percent and the Freight values by 3 percent for shippers in the United Kingdom:

UPDATE Orders

SET OrderAmount = OrderAmount \* 1.1,

Freight = Freight \* 1.03

WHERE ShipCountry = 'UK';

*If you want to know which records were updated, first examine the results of a select query that uses the same criteria, and then run the update query.*

## Create a delete query

- You can use a delete query to delete data from your tables, and you can use a delete query to enter criteria to specify which rows should be deleted. A delete query provides you an opportunity to review the rows that will be deleted before you perform the deletion.
- For example, say that while you were preparing to send the Chicago Orders table from the previous example, to your Chicago business associate, you notice that some of the rows contain a number of empty fields. You decided to remove these rows before you send the table. You could just open the table and delete the rows manually, but if you have many rows to delete and you have clear criteria for which rows should be deleted, you might find it helpful to use a delete query.
- You can use a query to delete rows in the Chicago Orders table that do not have a value for Order ID by using the following procedure:

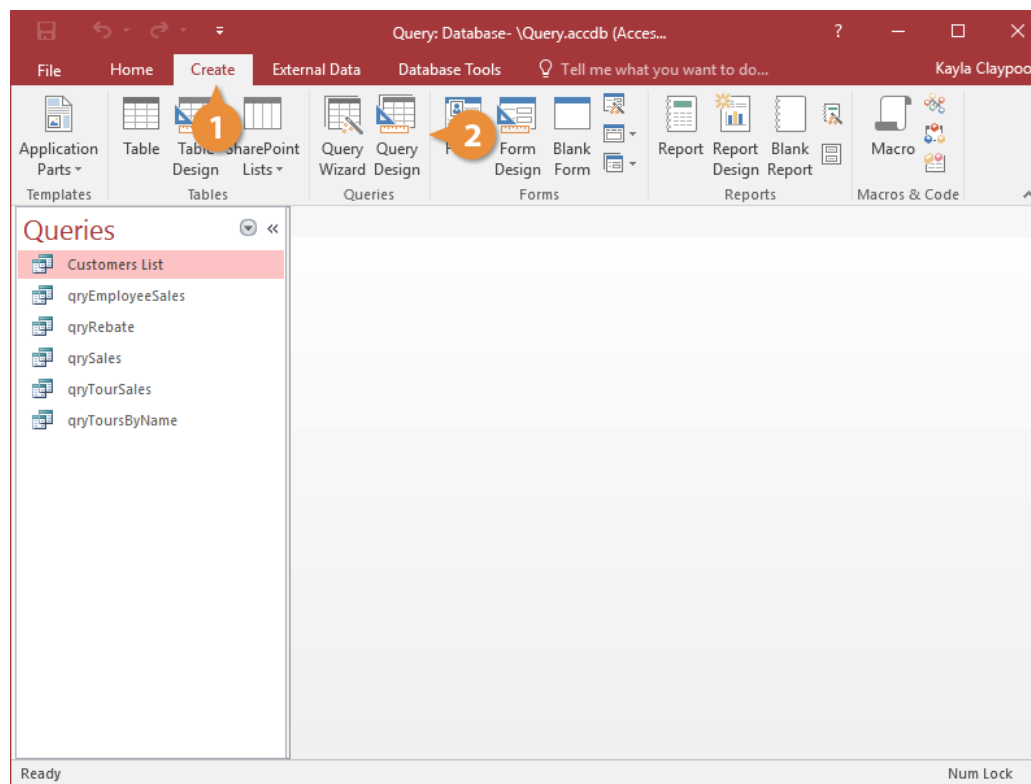
## Create a delete query

- You can use a query to delete rows in the Chicago Orders table that do not have a value for Order ID by using the following procedure:
  1. On the **Create** tab, click **Query Design**.
  2. Double-click **Chicago Orders**.
  3. On the **Design** tab, in the **Query Type** group, click **Delete**. In the design grid, the **Sort** and **Show** rows disappear, and the **Delete** row appears.
  4. In the **Chicago Orders** table, double-click **Order ID** to add it to the grid.
  5. In the design grid, in the **Criteria** row of the Order ID column, type **Is Null**.
  6. On the **Design** tab, in the **Results** group, click **Run**.



# Create a delete query

- Example:
  1. Click the **Create** tab on the ribbon.
  2. Click the **Query Design** button.

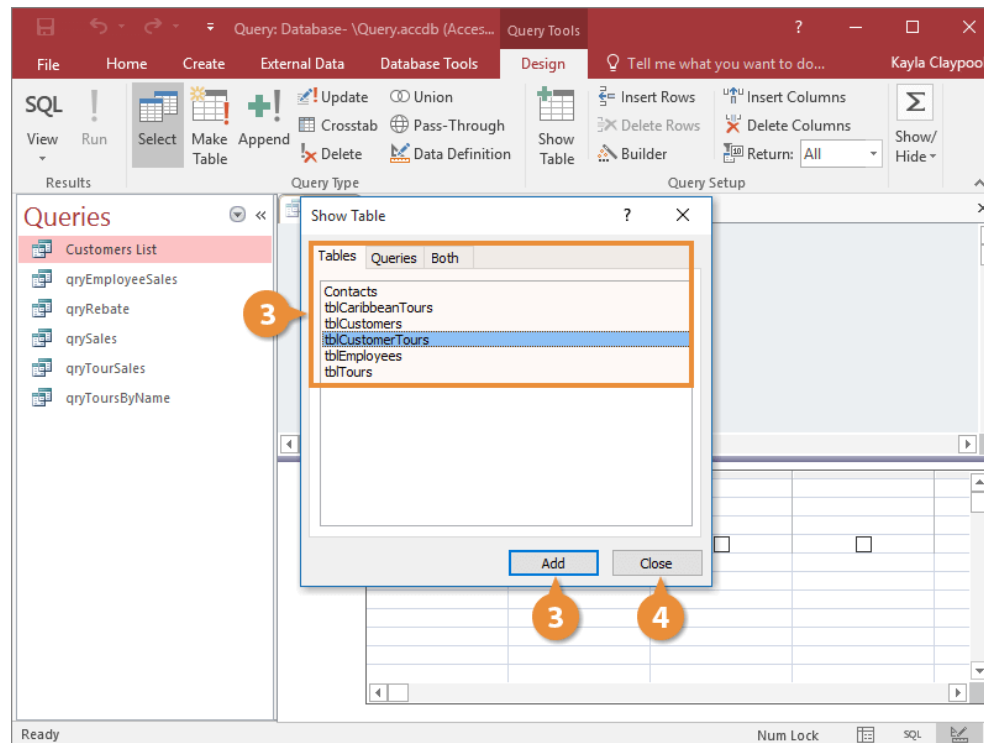


# Create a delete query

- Example:

The query design window and Show Table dialog box both appear. Now you need to select the tables and/or queries you want to use in the delete query.

3. Select the tables and queries you want to add and click **Add**.
4. Click **Close**.

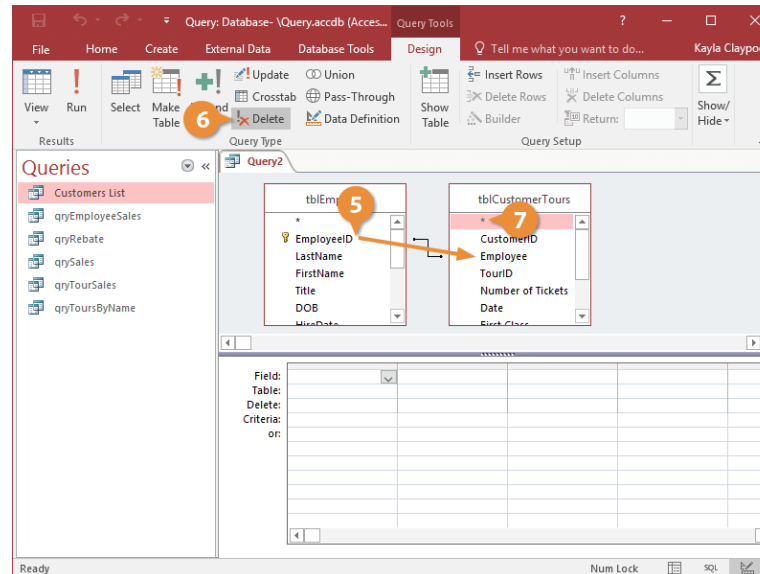


# Create a delete query

- Example:

If the tables are related, Access automatically connects their common fields with a join line. If the tables aren't related, you will have to manually join them by dragging a field from one table's field list to the matching field in the other table's field list.

5. Connect any unrelated tables. Next, tell Access that this is a Delete query.
6. Click the **Delete** button on the ribbon. Access converts the select query to a Delete query and displays the Delete row in the query design grid. Now you need to tell Access what you want to delete.
7. Double-click the **asterisk (\*)** from the table field list for the table from which you want to delete information.

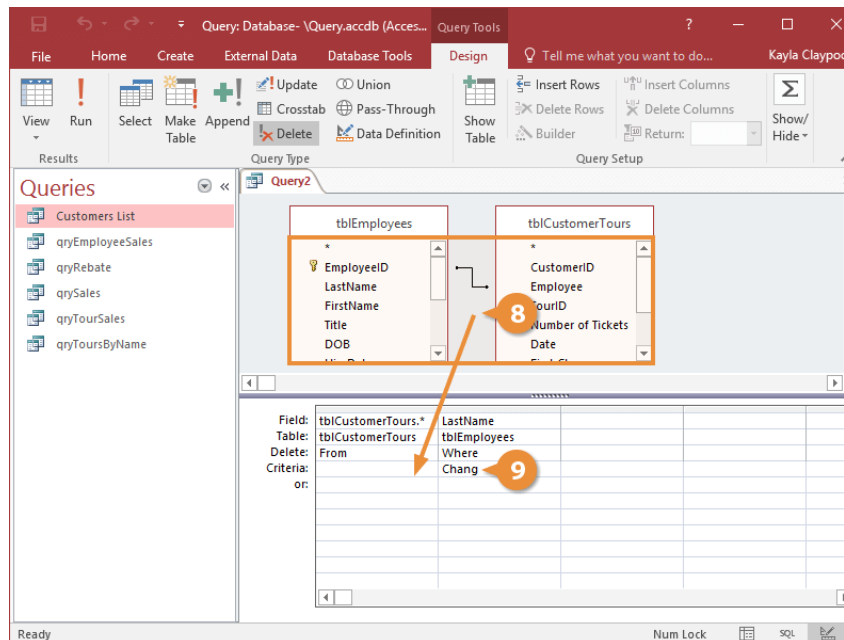


# Create a delete query

- Example:

Notice that From appears in the Delete cell for the asterisk field, indicating that the records will be deleted from this table. Unless you want the delete query to delete every record in the table, you will need to add some limiting criteria.

- Drag the field you want to use as the limiting criteria onto the design grid. Next you need to tell Access the specific data to delete.
- Click the field's **Criteria** row and type the specific data you want to delete and press **Tab**.



Access will add the quotation marks around the text string for you. That's all there is to creating a Delete query.

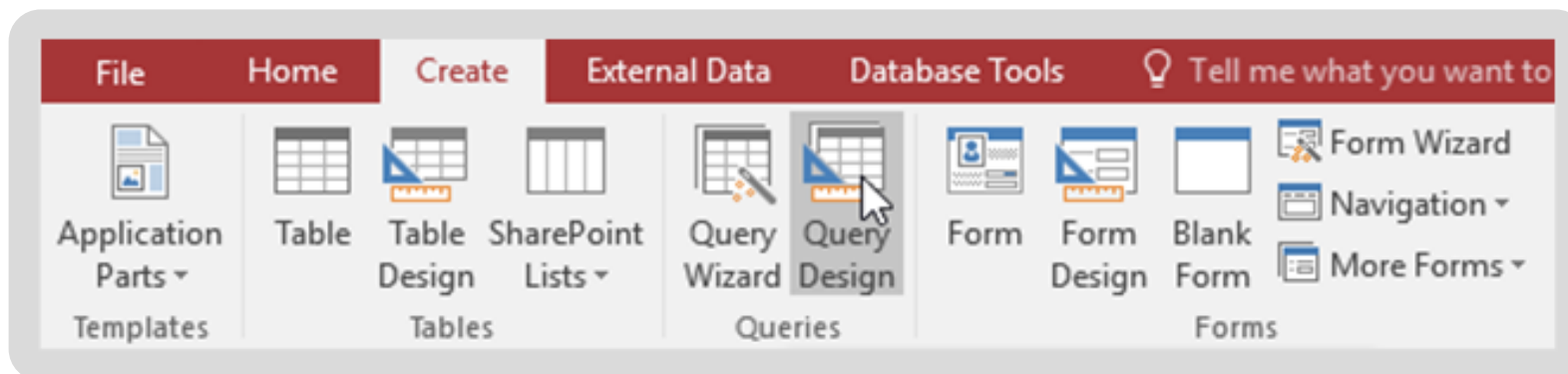
## Examples:

### One-table queries (By an Example)

- We intend to execute a query on the Customers table in our bakery database. The purpose of this query is to identify and compile a list of nearby customers who are most likely to attend a special event we are hosting at our bakery. The query will filter the Customers table to show only those customers who live in the nearby area. Consequently, we will obtain a list of relevant customers who fit our specific criteria.
- To locate customers who reside in Raleigh, we will conduct a search in the City field of our database. However, we also wish to invite customers who live in the suburbs that are in proximity to Raleigh. Therefore, we will include an additional criterion by adding the zip code 27513 to our search.
- It's possible that this approach may resemble the use of a filter, and indeed, a one-table query is essentially an advanced form of filtering applied to a single table.

## To create a simple one-table query:

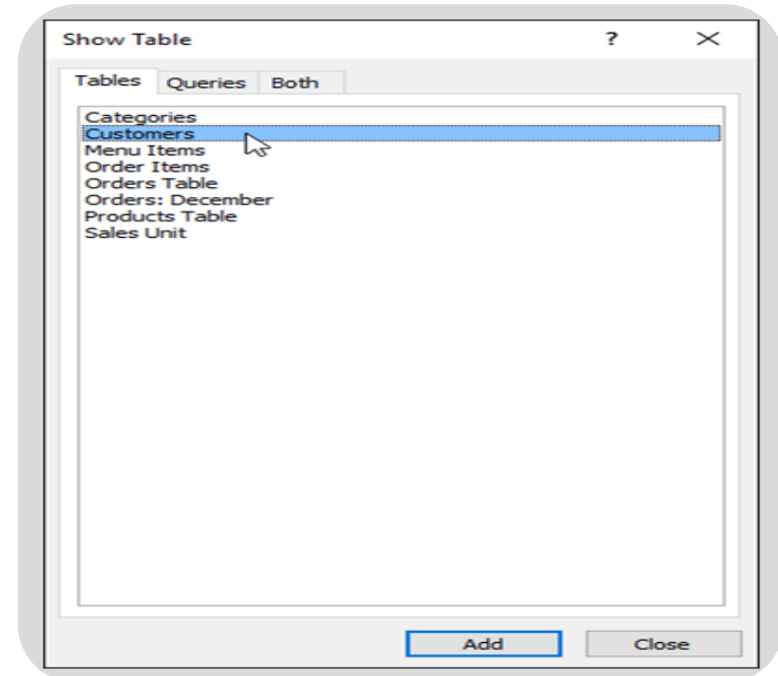
1. Select the **Create** tab on the Ribbon, and locate the **Queries** group.
2. Click the **Query Design** command.



## To create a simple one-table query:

3. Upon selecting the option to run a query, Access will switch to the Query Design view. A dialog box titled "Show Table" will appear, prompting the user to choose the table on which they wish to run the query. In this instance, we will select the Customers table, as that is the table we want to query.

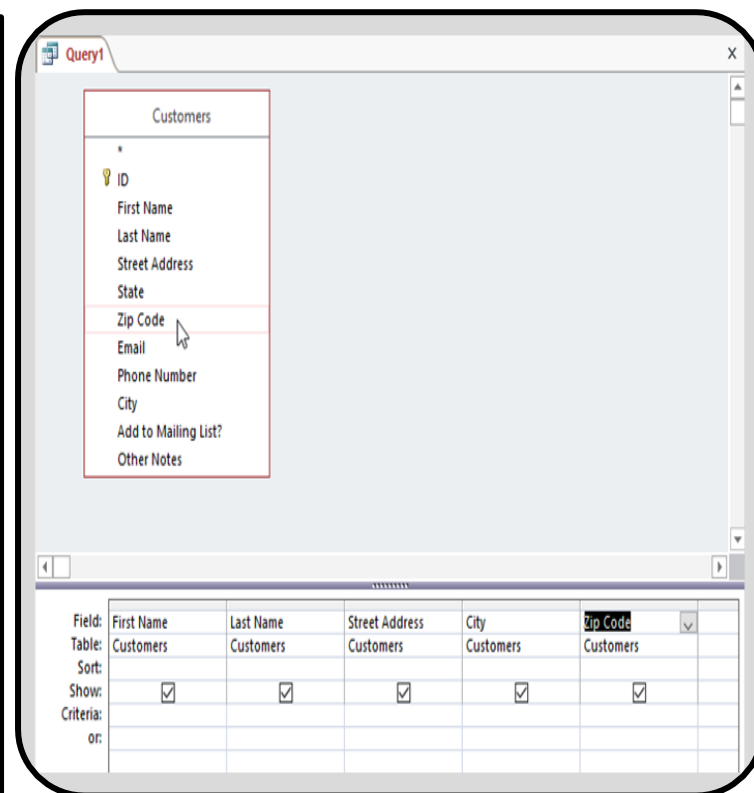
4. Click **Add**, then click **Close**.





## To create a simple one-table query:

5. Once the table is selected, it will be displayed as a small window in the Object Relationship pane. To add fields to the query, double-click on the desired field names within the table window. As each field is selected, it will be added to the design grid located at the bottom of the screen. For our particular query, we wish to send invitations to customers residing in a specific area. Therefore, we will include fields such as First Name, Last Name, Street Address, City, and Zip Code in our query design.



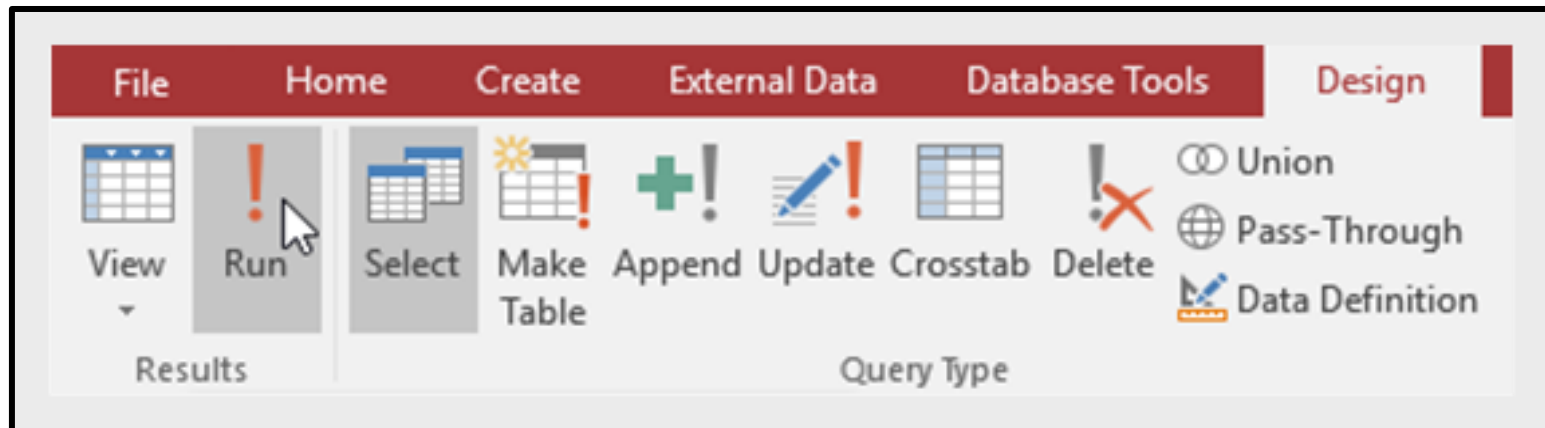
## To create a simple one-table query:

6. To establish the search criteria for the query, click on the cell located in the "Criteria" row for each field that needs to be filtered. If multiple criteria are entered into different fields within the "Criteria" row, the query will only display results that meet all specified criteria. However, if multiple criteria need to be set without requiring the records to meet all of them, the first criterion can be entered in the "Criteria" row, while the additional criteria can be entered in the "Or:" row and the rows below it. In this scenario, we want to locate customers who reside in Raleigh or have the 27513 zip code. Thus, we will enter "Raleigh" in the "Criteria" row for the City field and "27513" in the "Or:" row for the Zip Code field. By using quotation marks, the query will search for an exact match within these fields.

Field:	City	Zip Code	
Table:	Customers	Customers	
Sort:			
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Criteria:	'Raleigh'		
or:		'27513'	

## To create a simple one-table query:

7. Once the search criteria have been set, the query can be executed by clicking on the "Run" command, which is located on the Design tab.



8. The query output will be presented in the form of a table in the Datasheet view. If desired, the query can be saved by selecting the "Save" command located in the Quick Access Toolbar. After clicking on the "Save" command, a dialog box will appear, prompting the user to input a name for the query. Once the name has been entered, click on "OK" to save the query.

## To create a simple one-table query:

We have learned how to create a basic query that involves a single table.

Query1					
First Name	Last Name	Street Address	City	Zip Code	
Tracey	Beckham	7 East Walker Dr.	Raleigh	27612	
Lucinda	George	789 Brewer St.	Cary	27513	
Jerrold	Smith	211 St. George Ave.	Raleigh	27610	
Brett		h St.	Raleigh	27608	
Chloe			Raleigh	27609	
Alex			Cary	27513	
Nisha		h St.	Raleigh	27612	
Hillary			Raleigh	27606	
Katy	Jones	456 Denver Rd.	Cary	27513	
Beatrix	Joslin	85 North West St.	Raleigh	27606	
Mariah	Allen	12 Jupe	Raleigh	27605	
Jennifer	Hill	2100 Field Ave.	Raleigh	27609	
Cody	Hayes	65 North St.	Raleigh	27609	
Amaya	Gibson	5 West St.	Raleigh	27612	

Save As ? X

Query Name:

Nearby Customers I

OK Cancel

## **Designing a Multi-table Query:**

- It is highly likely that many of the queries you will create in Access will involve multiple tables, providing you the ability to address more intricate questions.
- If you are unclear about what you are looking for and how to achieve it, designing queries in Access can be a challenging task. While a basic query involving a single table may be straightforward enough to improvise, creating more sophisticated queries will require careful planning and consideration.

## Designing a Multi-table Query:

### Planning a Query

To prepare a query that involves multiple tables, you should follow these four steps:

1. Determine the specific question you want to answer using the query. While building a query is more complex than simply asking a question, having a clear idea of what you want to know is crucial to creating a useful query.
2. Identify all the types of information that you want to include in the query results. Which fields in the tables contain this information?
3. Locate the fields that you want to include in the query. Which tables do these fields belong to?
4. Determine the criteria that each field needs to meet. Consider the question you asked in step one. Which fields do you need to search to find specific information? What information are you looking for, and how will you search for it?

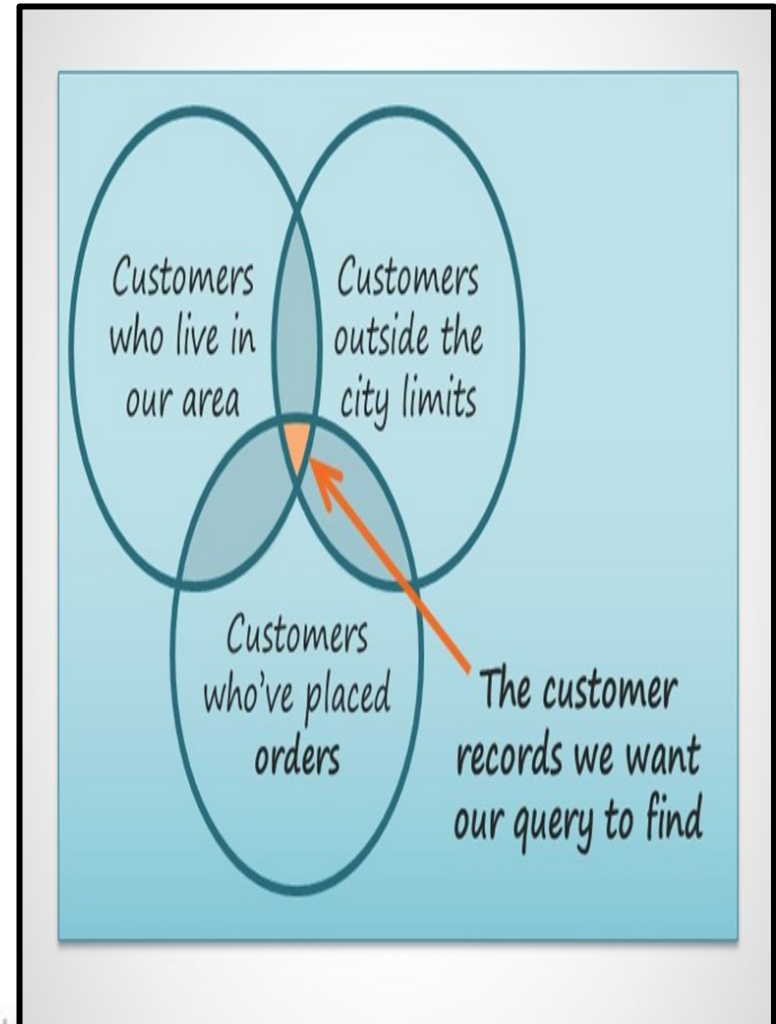


## Designing a Multi-table Query:

### Planning a Query

#### Step 1 : Pinpointing the question we want to ask

Our bakery database has a variety of customers, including some who have never ordered from us but are in our database because they subscribed to our mailing list. While most of our customers reside within our city, there are some who live outside of our area, even in different states. Our goal is to encourage our out-of-town customers who have placed orders in the past to visit us again by sending them some coupons. However, we only want to send coupons to those customers who live within our area and not too far away. Therefore, we need to identify customers who live outside our city but still reside within our area.





## Designing a Multi-table Query:

### Planning a Query

#### Step 2 :Identifying the information we need

In creating a list of customers, we need to consider the necessary information to include. This should cover the customers' personal details such as their names, contact information like addresses, phone numbers, and email addresses. Additionally, we want to identify customers who have previously placed orders. This can be done by including the order ID numbers in the list, allowing us to narrow down the list to customers who have previously placed orders.

*Information we  
need to answer  
our question*

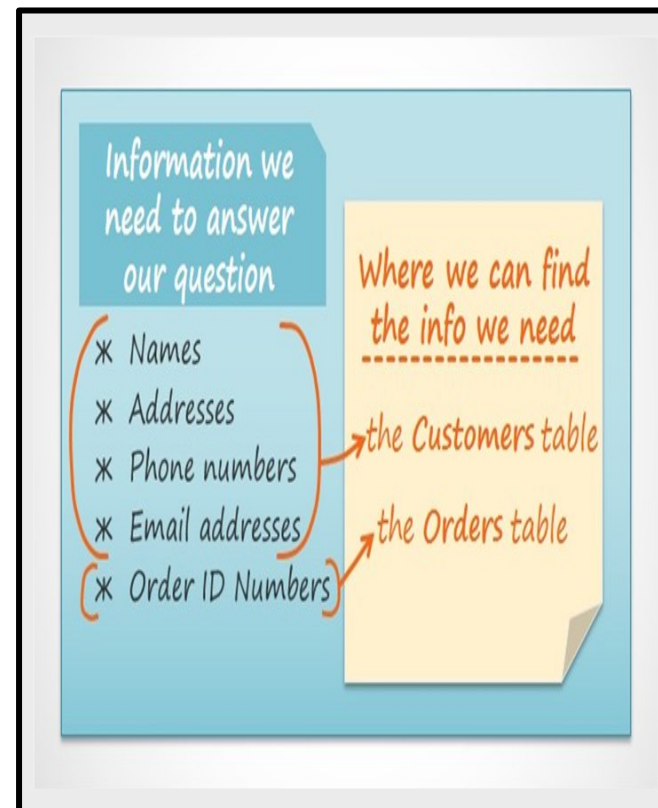
- \* Names
- \* Addresses
- \* Phone numbers
- \* Email addresses
- \* Order ID Numbers

## Designing a Multi-table Query:

### Planning a Query

#### Step 3 :Finding the tables that have the data we require.

To create a query, it's important to have a good understanding of the tables in your database. In our case, we are familiar with our own database and we know that the relevant customer information is stored in the Customers table. The Order ID numbers we need are located in the Orders table. By including these two tables in our query, we can retrieve all the necessary information.



## Designing a Multi-table Query:

### Planning a Query

#### Step 4 : Determining the criteria our query for search

To answer our question, we need to set criteria that filter out customers who live within our city but include those who live in our area outside the city limits. We can set criteria for the "City" field to exclude our city name, and we can set criteria for the "State" field to include only our state abbreviation. This will give us a list of customers who live in our area but not within our city limits. For the "Order ID" field, we can set criteria to retrieve only those customers who have placed orders in the past.

*Criteria the query should use to find customer records:*

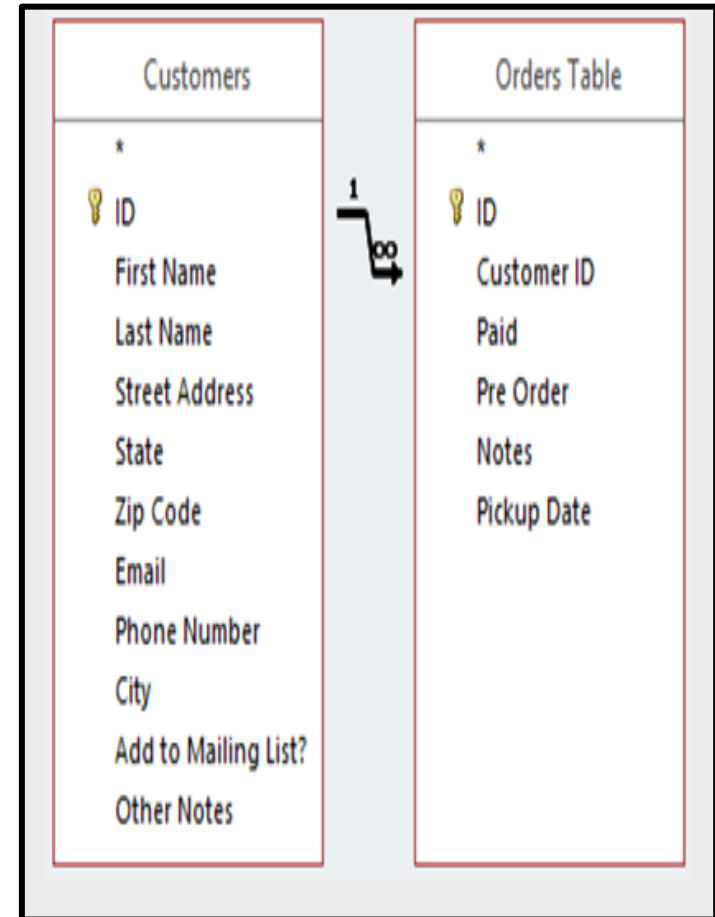
- ✱ No one living in our town, Raleigh
  - ✱ In the City field, type **Not In** ("Raleigh")
- ✱ Only customers with phone numbers that start with "919"
  - (So we only get customers who live nearby)
  - ✱ In the Phone Number field, type **Like** ("919\*")

## Designing a Multi-table Query:

### Joining tables in queries

When creating a query in Access with multiple tables, it's important to consider how to link or join the tables. In the Object Relationship pane, you will see how the two tables are related to each other.

The line connecting the two tables in the Object Relationship pane is referred to as the join line. It is an arrow-shaped line that indicates the sequence in which the query processes data from the two tables. In the given image, the arrow is pointing from left to right, which means the query will first examine the data in the left table and then look at only the relevant data in the right table that corresponds to the records it has already examined in the left table.

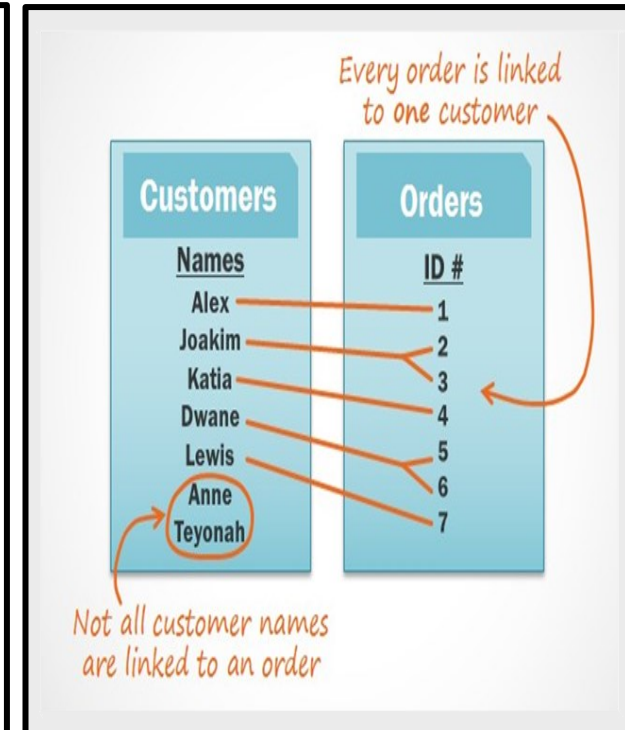


## Designing a Multi-table Query:

### Joining tables in queries

To ensure that your query retrieves the correct information, you may need to change the direction of the join between tables. The join direction can affect the information your query retrieves. Sometimes, Access will join tables from right to left instead of left to right, depending on the relationship between the tables.

In our query, we have included the Customers table and the Orders table to see customers who have placed orders. To understand this better, let's examine the data in these tables.

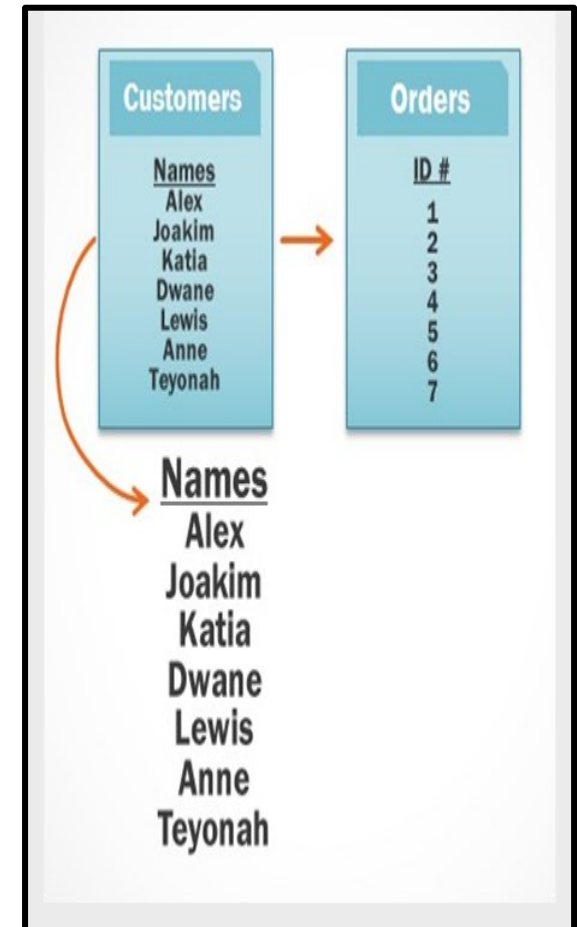




## Designing a Multi-table Query:

### Joining tables in queries

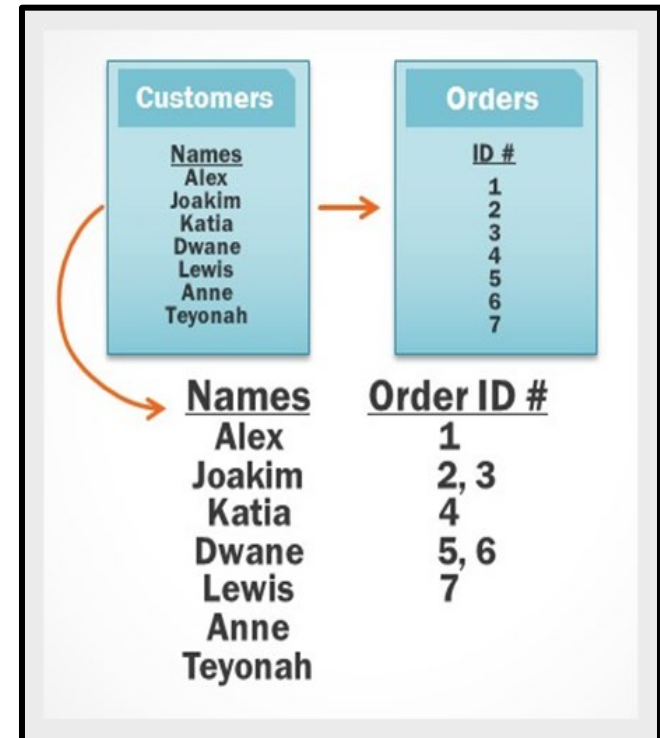
- When examining the two tables, it becomes clear that every order in the Orders table is associated with a customer in the Customers table.
- However, some customers in the Customers table have never placed an order and are not linked to any orders, while others who have placed multiple orders are connected to more than one order.
- Therefore, even when two tables are linked, it is possible for one table to contain records that have no relationship with any record in the other table.
- If Access tries to execute our query with the current join direction, from left to right, it will retrieve all records from the left table, which is our Customers table.



## Designing a Multi-table Query:

### Joining tables in queries

After retrieving every record from the left table, Access will then fetch every record from the right table that is related to the records retrieved from the left table. Our current join, which starts with the Customers table, includes all customers regardless of whether they've placed orders or not. This results in more information than we require. We only want to retrieve records for customers who have placed orders.

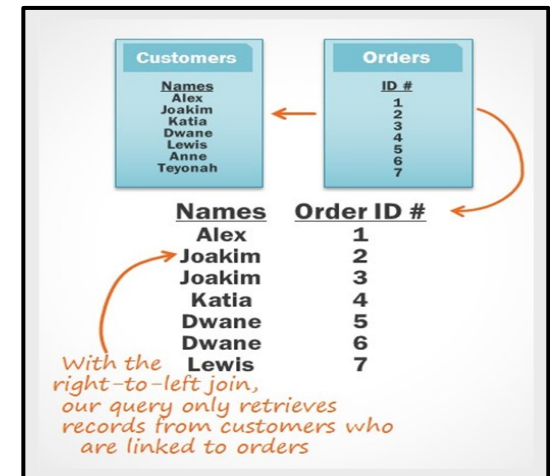
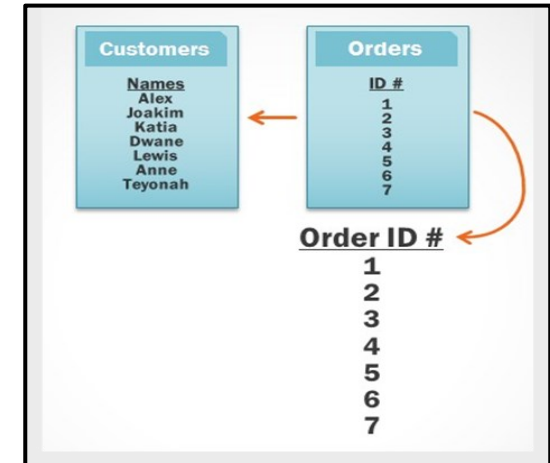




## Designing a Multi-table Query:

### Joining tables in queries

- We can solve this issue by simply changing the direction of the join line. If we join the tables from right to left instead, Access will start by retrieving the orders from the Orders table:
- By changing the direction of the join line to right-to-left, Access will first retrieve the orders from the Orders table, and then look at the Customers table to retrieve only the records of customers who are linked to an order in the Orders table. This will ensure that our query only includes records for customers who have placed orders, which is what we need.

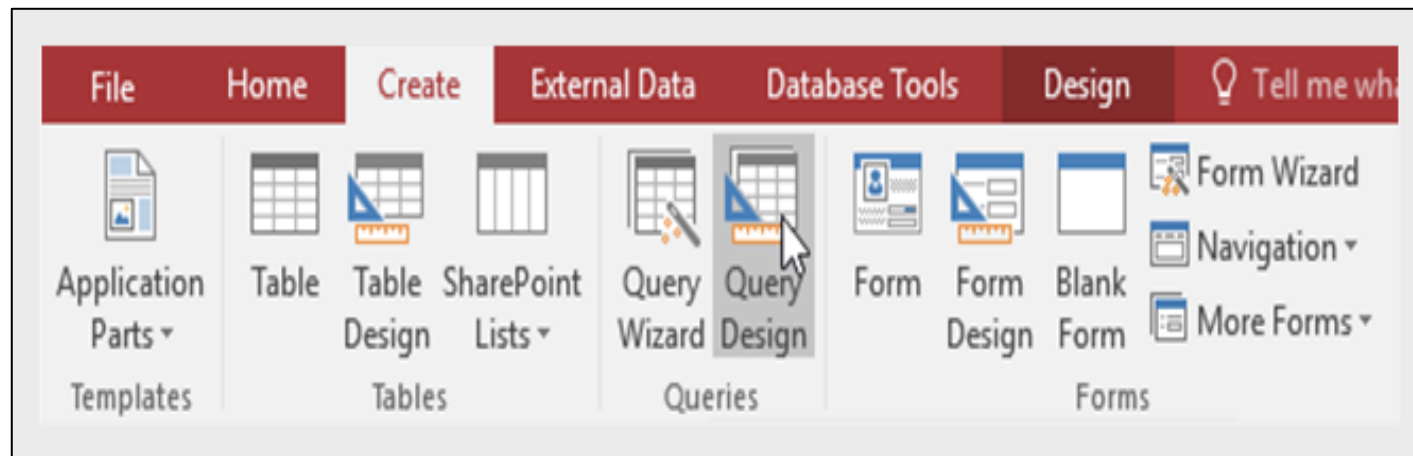


## Creating a multi-table query

- Once we have completed the planning process for our query, we can begin designing and executing it. It is important to refer to any written plans we have created during the query design process.

### To create a multi-table query:

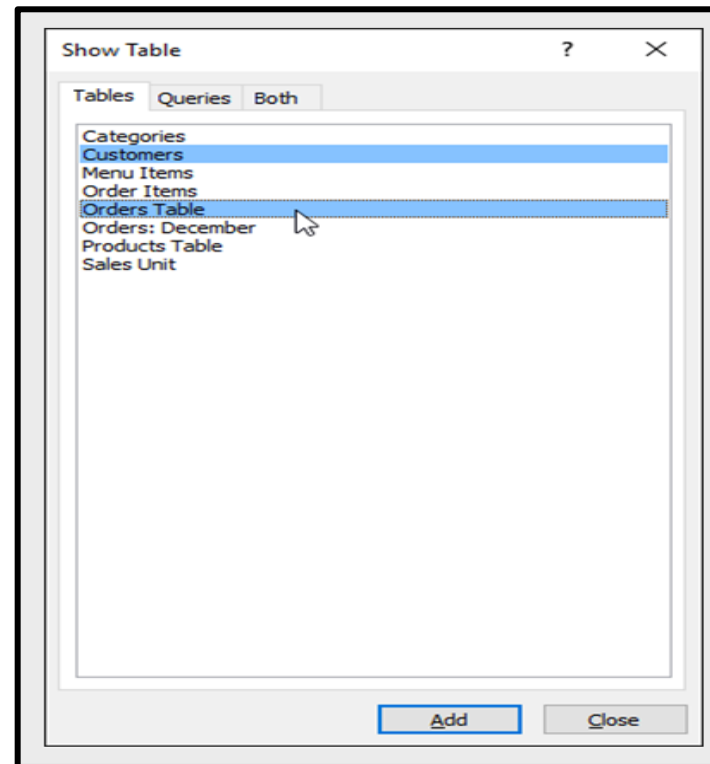
- Select the Query Design command from the Create tab on the Ribbon



## Creating a multi-table query

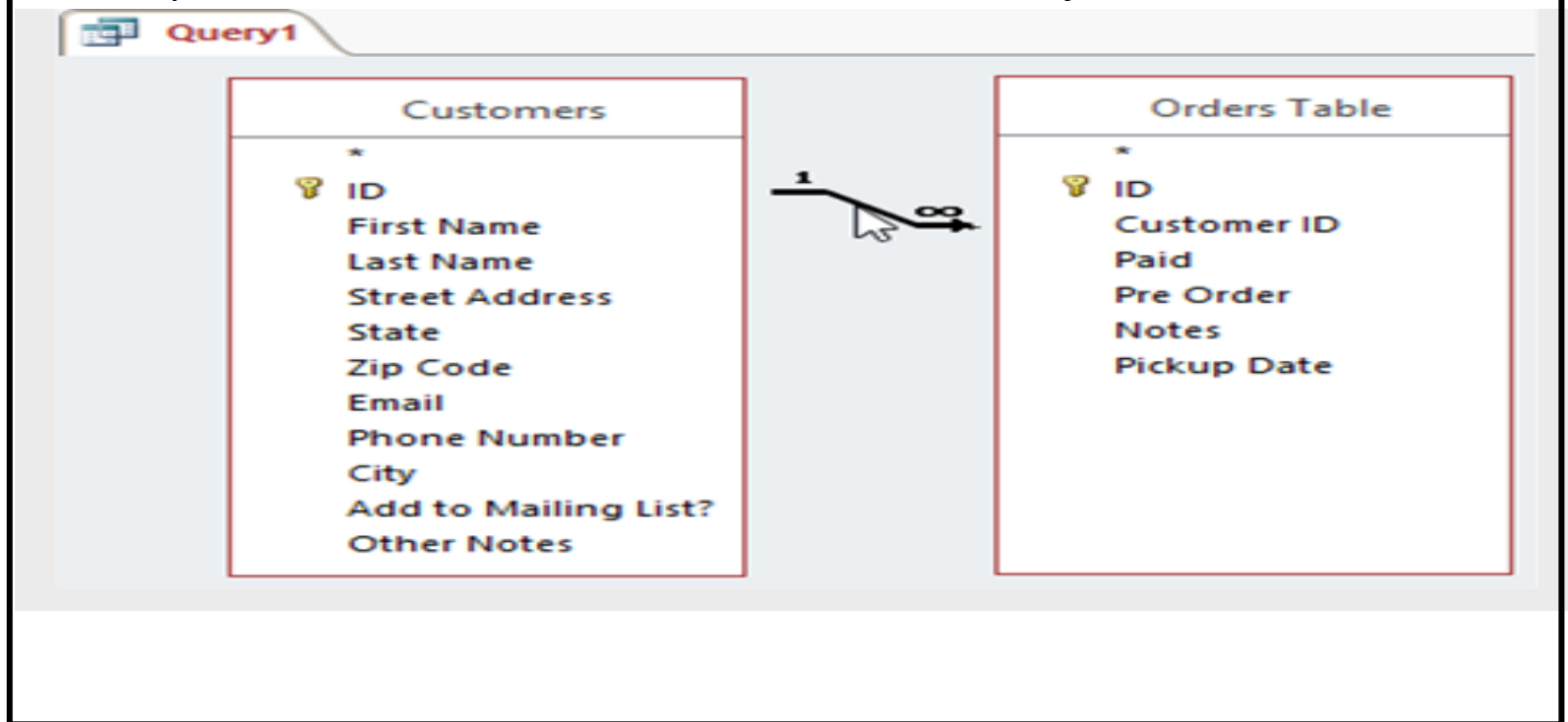
2. To include tables in your query, you need to select them in the dialog box that appears and click the "Add" button. You can select multiple tables by holding down the Ctrl key on your keyboard. In our case, we decided that we needed data from the Customers and Orders tables when we planned our query, so we will add these tables.

3. After you have added all of the tables you want, click Close.



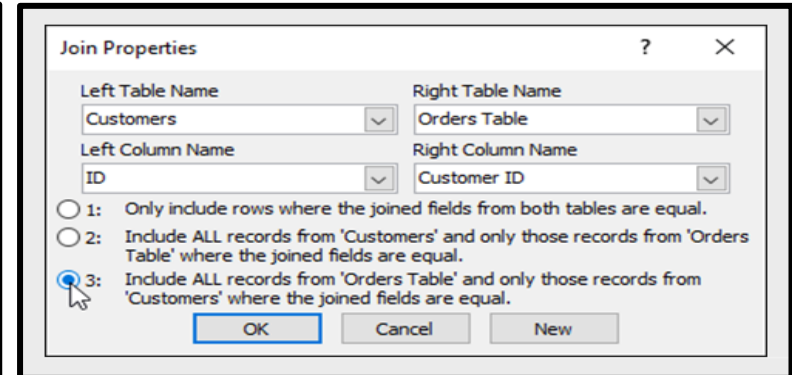
## Creating a multi-table query

4. You will see the tables you added in the Object Relationship pane, connected by a join line. If you want to edit the join direction between two tables, you can double-click on the thin section of the join line.



## Creating a multi-table query

5. To change the join direction, you can double-click on the join line to open the Join Properties dialog box. In this dialog box, you can select the option that specifies the direction of your join. For instance, if you want a right-to-left join, you can choose option 3.

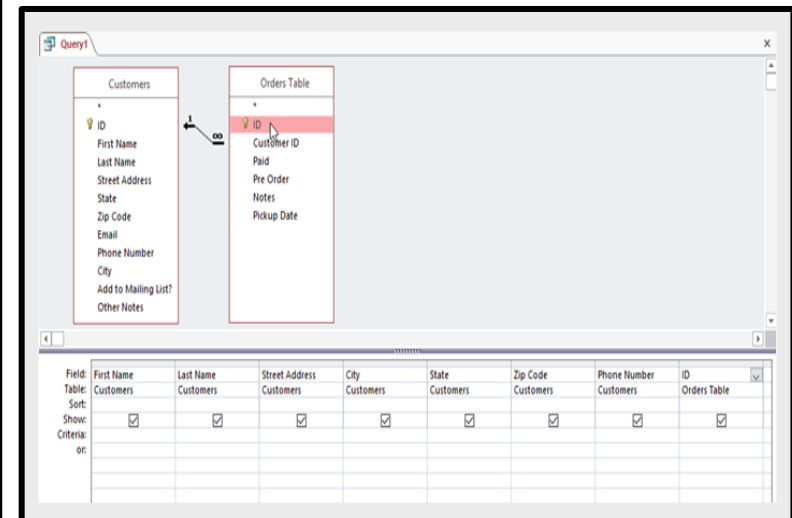


The Join Properties dialog box shows the following settings:

- Left Table Name: Customers
- Right Table Name: Orders Table
- Left Column Name: ID
- Right Column Name: Customer ID
- Option 3 is selected: Include ALL records from 'Orders Table' and only those records from 'Customers' where the joined fields are equal.

Buttons: OK, Cancel, New

6. To add fields to your query, double-click the desired field names in the table windows. These fields will be added to the design grid located in the bottom part of the screen. For example, in our scenario, we will add several fields from the Customers table such as First Name, Last Name, Street Address, City, State, Zip Code, and Phone Number. We will also add the ID number from the Orders table.



The Query1 design grid shows the following fields added to the query:

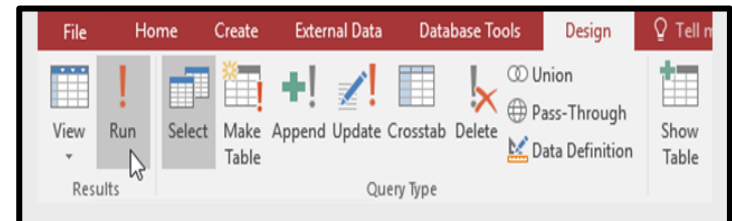
Field	First Name	Last Name	Street Address	City	State	Zip Code	Phone Number	ID
Table:	Customers	Customers	Customers	Customers	Customers	Customers	Customers	Orders Table
Sort:								
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:								
or:								

## Creating a multi-table query

7. To set field criteria, type the desired criteria in the criteria row of each field. For our example query, we want to set two criteria: "Not in ('Raleigh')" in the City field and "Like ('919\*')" in the Phone Number field. This will help us find customers who don't live in Raleigh but live in the 919 area code.

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

Field:	City	State	Zip Code	Phone Number	ID
Table:	Customers	Customers	Customers	Customers	Orders Table
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:	Not In ('Raleigh')			Like ('919*')	
or:					



## Creating a multi-table query

9. The query results will be presented in the Datasheet view, which resembles a table. You can save your query by clicking the Save command in the Quick Access Toolbar and entering the desired name when prompted, then clicking OK.

**Creating a multi-table query is completed**

Last Name	Street Address	City	State
Williams	9014 Miller Ln.	Durham	NC
Daugherty	105 Aycock St.	Chapel Hill	NC
Olsen	4325 W. King St.	Garner	NC
Sigrudsdatter			NC
Yuen			NC
MacDonald			NC
Slobodowski			NC
Oglesby			NC
Kellerman			NC
Olivera	60 Glenwood Ave Apt A121	Durham	NC
Storey	1834 Dakota St.	Durham	NC
Tempie	12 Spencer Ave.	Chapel Hill	NC
Emory	99 Hillsborough St.	Garner	NC

Save As ? X

Query Name:

Customers Who've Ordered from Nearby Towns

OK Cancel

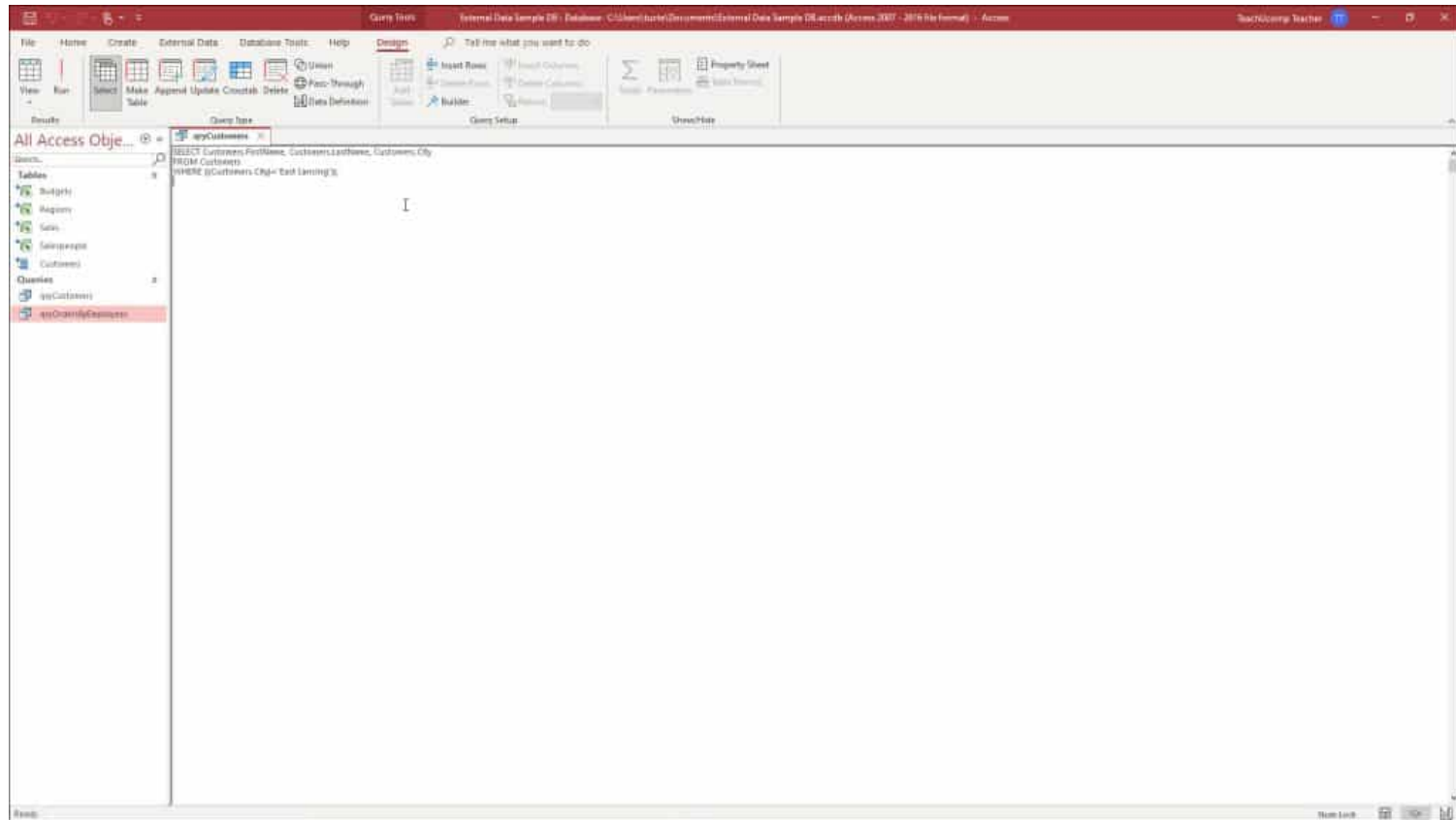


## SQL View in Access

- SQL view in Access lets you see the SQL code of Access queries. When you are visually creating the query in the query design view in Access, what you are really doing is visually constructing SQL code. SQL stands for Structured Query Language.
- If you want to learn SQL, a good way to start is by viewing the SQL code your Access queries produce. To view the SQL code for an Access query, open the query in query design view. Then click the “View” drop-down button in the “Results” button group on the “Design” tab of the “Query Tools” contextual tab in the Ribbon. From the drop-down menu of choices that appears, select the “SQL View” command. Access then shows your query as SQL code, so you can see how it works.
- If you are familiar with how SQL is constructed in Access, you can also create and edit the SQL code directly in the SQL view, if desired. If you do makes changes to the query’s SQL in this view you want to save, make sure to click the “Save” button in the Quick Access toolbar.

# SQL View in Access

- SQL view in Access:



# SQL statements

- The following table shows sample SQL statements that employ an expression:

SQL statement that uses an expression	Result
<code>SELECT [FirstName],[LastName] FROM [Employees] WHERE [LastName]="Danseglio";</code>	Displays the values in the FirstName and LastName fields for employees whose last name is Danseglio.
<code>SELECT [ProductID],[ProductName] FROM [Products] WHERE [CategoryID]=Forms![New Products]![CategoryID];</code>	Displays the values in the ProductID and ProductName fields in the Products table for records in which the CategoryID value matches the CategoryID value specified in an open New Products form.
<code>SELECT Avg([ExtendedPrice]) AS [Average Extended Price] FROM [Order Details Extended] WHERE [ExtendedPrice]&gt;1000;</code>	Calculates the average extended price for orders for which the value in the ExtendedPrice field is more than 1000, and displays it in a field named Average Extended Price.
<code>SELECT [CategoryID], Count([ProductID]) AS [CountOfProductID] FROM [Products] GROUP BY [CategoryID] HAVING Count([ProductID])&gt;10;</code>	In a field named CountOfProductID, displays the total number of products for categories with more than 10 pro

# SQL statements

- Creates an update query that changes values in fields in a specified table based on specified criteria:

**UPDATE** *table*  
**SET** *newvalue*  
**WHERE** *criteria*;

- The UPDATE statement has these parts:

Part	Description
<i>table</i>	The name of the table containing the data you want to modify.
<i>newvalue</i>	An expression that determines the value to be inserted into a particular field in the updated records.
<i>criteria</i>	An expression that determines which records will be updated. Only records that satisfy the expression are updated.

## SQL statements

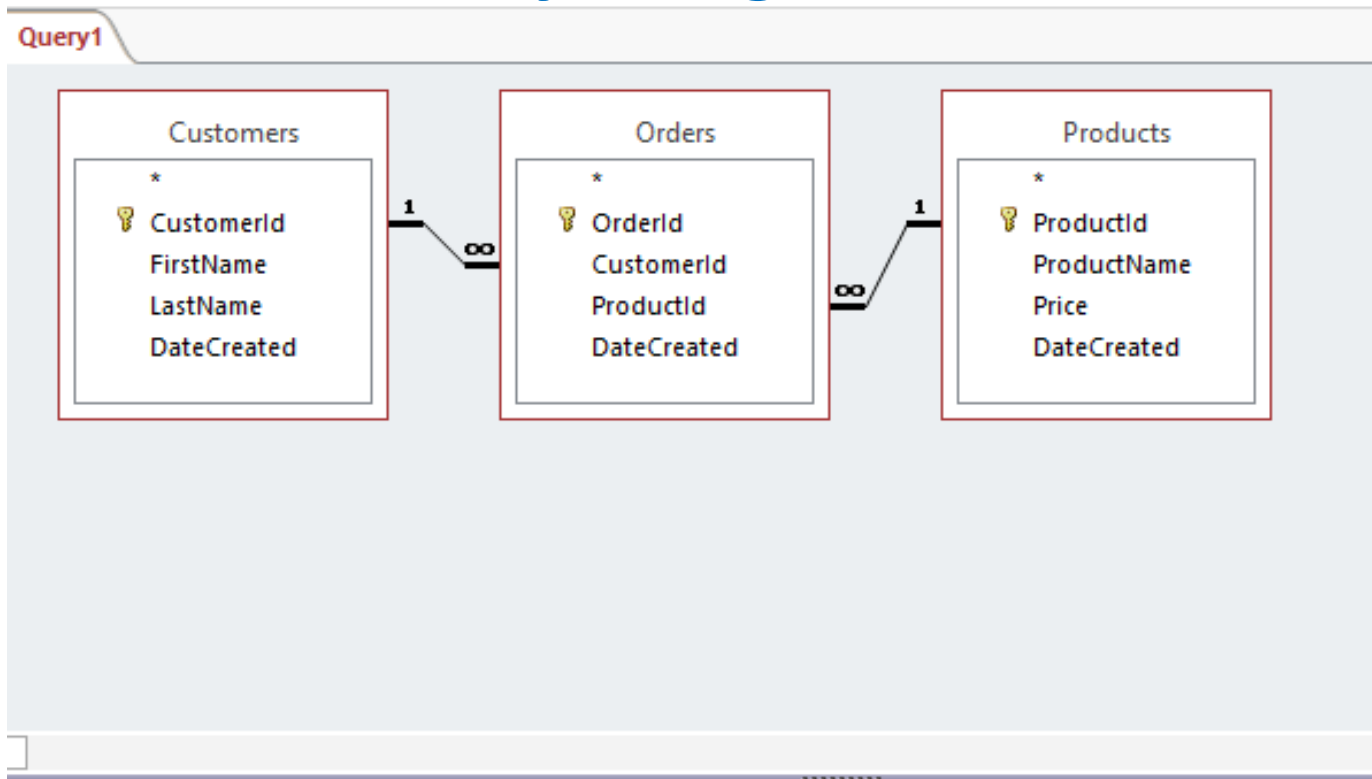
- Creates a delete query that removes records from one or more of the tables listed in the FROM clause that satisfy the WHERE clause:

**DELETE** [*table.\**]  
**FROM** *table*  
**WHERE** *criteria*

- The DELETE statement has these parts:

Part	Description
<i>table</i>	The optional name of the table from which records are deleted.
<i>table</i>	The name of the table from which records are deleted.
<i>criteria</i>	An expression that determines which records to delete.

# Query Design View



Field:	CustomerId	FirstName	LastName	ProductName	Price
Table:	Customers	Customers	Customers	Products	Products
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:					
or:					

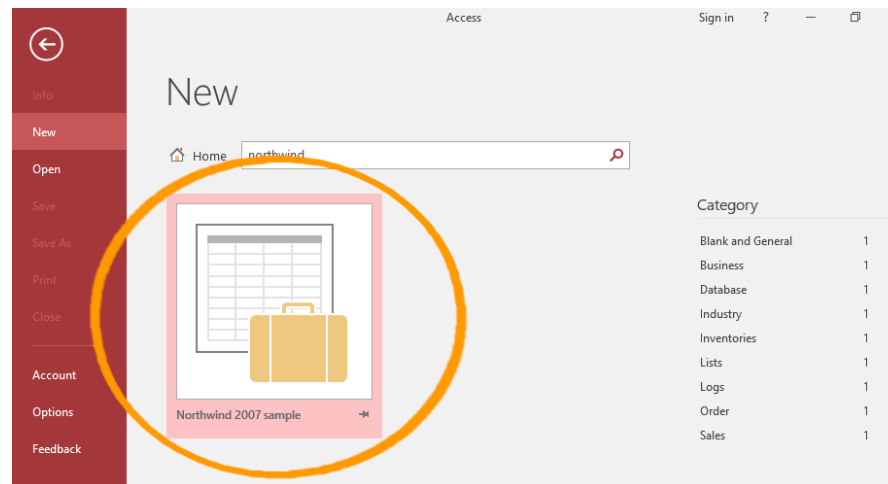
# The Query Results

Query1				
CustomerId	FirstName	LastName	ProductName	Price
1	Homer	Simpson	Venus Carrera ET	\$190,000.99
2	Peter	Griffin	Mars Dreamliner 787	\$82,000.00
10	Bender	Rodríguez	Mercury Riser 2020	\$55,000.00
10	Bender	Rodríguez	Pluto Mini Racer	\$25,000.00
3	Stewie	Griffin	Mars Daytripper	\$35,000.00
4	Brian	Griffin	Venus Carrera ET	\$190,000.99
5	Cosmo	Kramer	Mercury Riser 2020	\$55,000.00
6	Philip	Fry	Saturn SUV	\$65,750.00
7	Amy	Wong	Pluto Mini Racer	\$25,000.00
8	Hubert J.	Farnsworth	Mars Dreamliner 787	\$82,000.00
1	Homer	Simpson	Saturn SUV	\$65,750.00
6	Philip	Fry	Venus Carrera ET	\$190,000.99
*	(New)			



# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



Using this database, please try to create different queries.



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

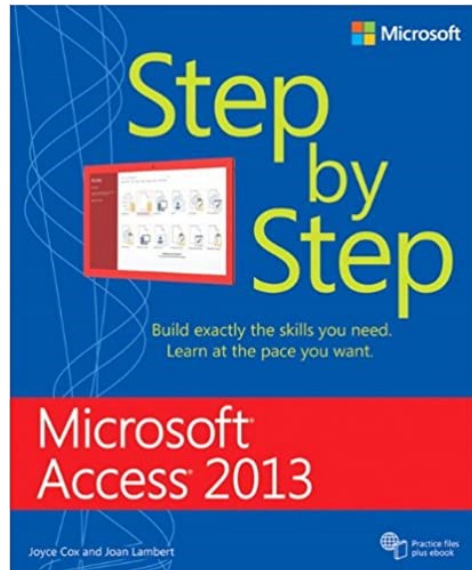
## ☐ Module 5. Practice

### ☐ Topic 3. Creation of a query. Use of queries.

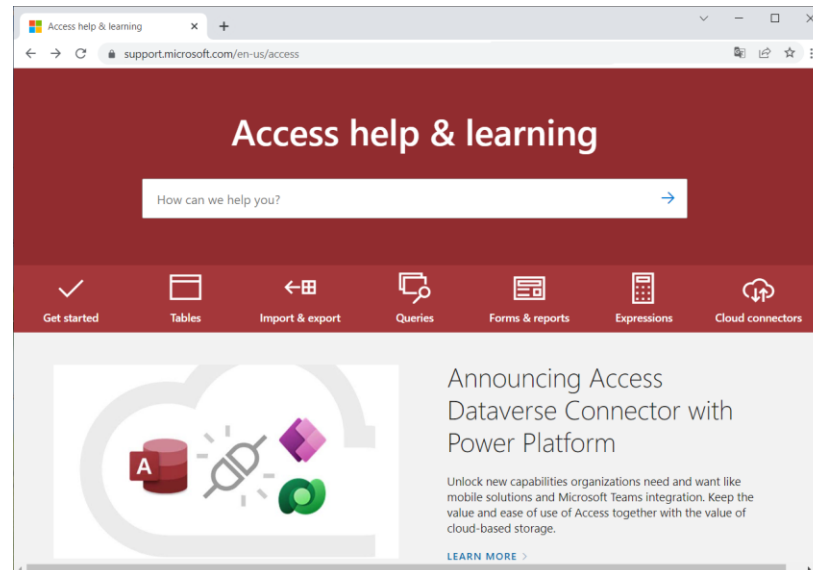
#### ☐ Practical lesson 2. Use of queries



# Practical lesson. Use of queries.



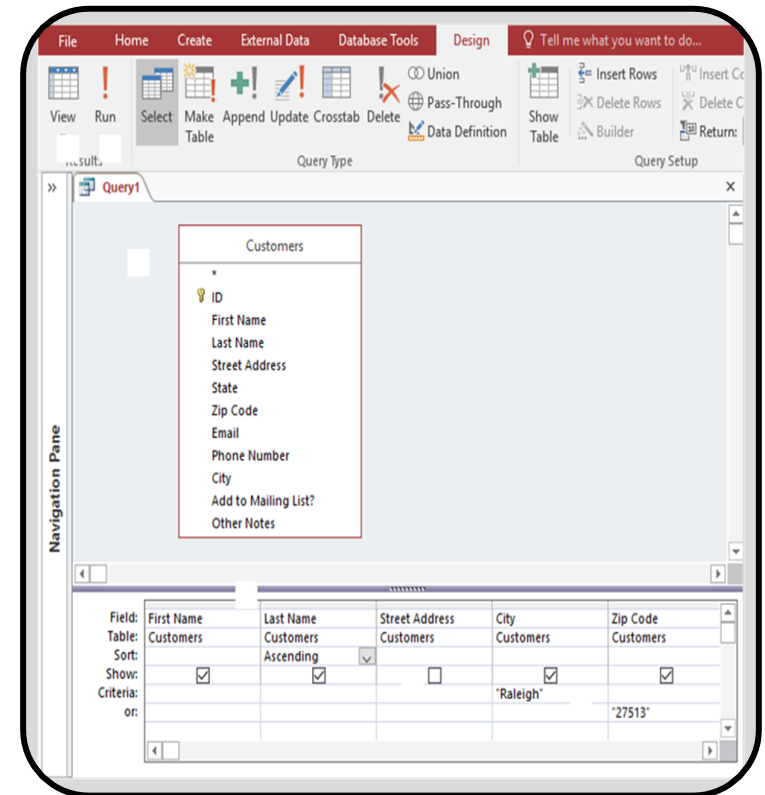
Joyce Cox, Joan Lambert.  
*Microsoft Access 2013 Step By Step*, Microsoft Press., 2013.



<https://support.microsoft.com/en-us/access>  
<https://www.customguide.com>

## How are queries used?

- The strength of queries exceeds that of basic searches or filters used to locate data within a table. This is due to queries being able to gather information from various tables simultaneously. Unlike searches that can only look in one table, queries can draw data from several sources, allowing for more comprehensive and complex analysis.
- Although query results are typically displayed in a table format, the process of designing a query involves a different perspective. Rather than focusing on the presentation of data, query design involves constructing a blueprint of how the data will be retrieved and combined from multiple tables. The resulting query is then used to generate a table displaying the desired information.



## Run a query

- A query is a set of instructions that you can use for working with data. You run a query to perform these instructions. In addition to returning results — which can be sorted, grouped, or filtered — a query can also create, copy, delete, or change data.
- This lesson explains how to run queries and provides only brief overviews of the various types of queries. The lesson also discusses error messages you might encounter when you run different types of queries, and provides steps you can take to work around or correct those errors.

## Run a query

- Important: You cannot run action queries if a database is operating in Disabled mode —a reduced functionality mode that Access uses to help protect your data in certain circumstances. You may see a dialog box warning, or you may see a warning in the Message Bar
- **Run a select or a crosstab query:**
  - You use select queries and crosstab queries to retrieve and present data, and to supply forms and reports with data. When you run a select or a crosstab query, Access displays the results in Datasheet view.



## Run a query

- **Run the query**
  1. Locate the query in the Navigation Pane.
  2. Do one of the following:
    1. Double-click the query you want to run.
    2. Click the query you want to run, and then press ENTER.

If the query you want to run is currently open in Design view, you can also run it by clicking **Run** in the **Results** group on the **Design** tab on the Ribbon, part of the Microsoft Office Fluent user interface.

## Run a query

- **Run an action query**
  - There are four types of action queries: append queries, delete queries, update queries, and make-table queries.
  - Except for make-table queries (which create new tables), action queries make changes to the data in tables they are based on.
  - These changes cannot be easily undone, for example, by pressing CTRL+Z. If you make changes using an action query that you later decide you didn't want to make, usually you will have to restore the data from a backup copy.
  - For this reason, you should always make sure you have a fresh backup of the underlying data before running an action query.

## Run a query

- **Run an action query**
  - You can mitigate the risk of running an action query by first previewing the data that will be acted upon. There are two ways to do this:
    - View the action query in Datasheet view before you run it. To do this, open the query in Design view, click **View** on the Access status bar, and then click **Datasheet View** on the shortcut menu. To switch back to Design view, click **View** again, and then click **Design View** on the shortcut menu.
    - Change the query to a select query, and then run it.

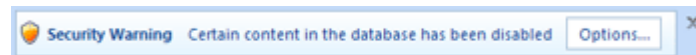
Note: Make sure to note what type of action query (append, update, make-table, or delete) you are starting with, so you can change the query back to that type after you preview the data with this method.

## Run a query

- **Run an action query**
  - Run an action query as a select query:
    - a. Open the action query in Design view.
    - b. On the **Design** tab, in the **Query Type** group, click **Select**.
    - c. On the **Design** tab, in the **Results** group, click **Run**.
  - Run the query

When you are ready to run an action query, double-click it in the Navigation Pane, or click it and then press ENTER.

Important: By default, Access disables all action queries in a database unless you indicate that you trust the database. You can indicate that you trust a database by using the Message Bar, just below the Ribbon.



## Run a query

- **Run an action query**
  - Trust a database
    1. On the Message Bar, click **Options**.  
The **Microsoft Office Security Options** dialog box appears.
    2. Select **Enable this content** and then click **OK**.
- **Run a parameter query**
  - A parameter query prompts you for a value when you run it. When you supply the value, the parameter query applies it as a field criterion. Which field it applies the criterion to is specified in the query design. If you do not supply a value when prompted, the parameter query interprets your input as an empty string.

## Run a query

- **Run a parameter query**
  - A parameter query prompts you for a value when you run it. When you supply the value, the parameter query applies it as a field criterion. Which field it applies the criterion to is specified in the query design. If you do not supply a value when prompted, the parameter query interprets your input as an empty string.
  - A parameter query is always also another type of query. Most parameter queries are select queries or crosstab queries, but append, make-table, and update queries can also be parameter queries.
  - You run a parameter query according to its other query type, but, in general, use the following procedure.

## Run a query

- **Run a parameter query**

- Run the query

1. Locate the query in the Navigation Pane.

2. Do one of the following:

1. Double-click the query you want to run.

2. Click the query you want to run, then press ENTER.

3. When the parameter prompt appears, enter a value to apply as a criterion.



## Run a query

- **Run a SQL-specific query**
  - There are three main types of SQL-specific query: union queries, pass-through queries, and data-definition queries.
  - Union queries combine data from two or more tables, but not in the same manner as other queries. Whereas most queries combine data by concatenating rows, union queries combine data by appending rows.
  - Union queries differ from append queries in that union queries do not change the underlying tables. Union queries append the rows in a recordset that does not persist after the query is closed.

## Run a query

- **Run a SQL-specific query**
  - Pass-through queries are not processed by the database engine that comes with Access; rather, they are passed directly to a remote database server that does the processing and then passes the results back to Access.
  - Data-definition queries are a special type of query that does not process data; instead, data-definition queries create, delete or modify other database objects.
  - SQL-specific queries cannot be opened in Design view. They can only be opened in SQL view, or run. Except for data-definition queries, running a SQL-specific query opens it in Datasheet view.

## Run a query

- **Run a SQL-specific query**

- Run the query

1. Locate the query in the Navigation Pane.

2. Do one of the following:

1. Double-click the query you want to run.

2. Click the query you want to run, and then press ENTER.

- **Troubleshoot an error message**

The following table shows some common error messages you may encounter. These errors can appear either as a message in a cell (instead of an expected value), or as an error message. The sections that follow the list include procedures you can use to resolve these errors.

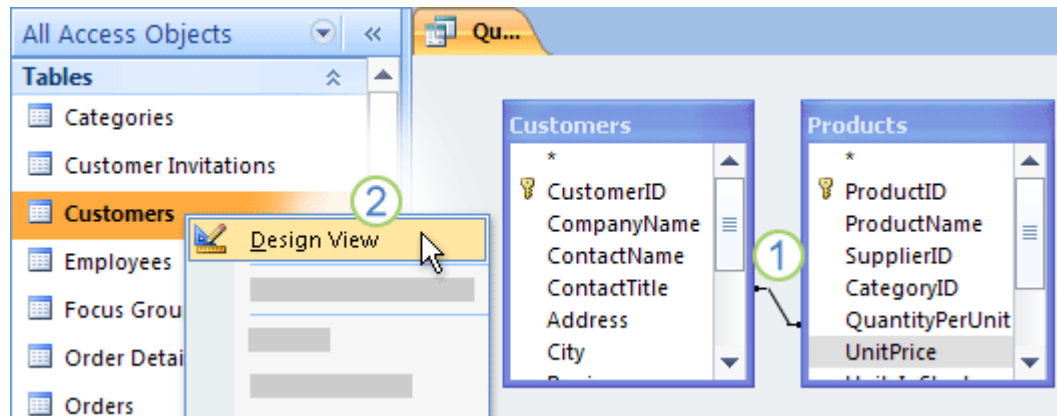
# Run a query

- Run a SQL-specific query

Error message	Problem	Solution
<b>Type mismatch in expression</b>	The query may be joining fields that have different data types.	Check the query design and ensure that the joined fields have the same data type. For instructions, see the section.
<b>Record is Deleted</b>	This can occur if either the object or the database is damaged.	Compact and repair the database. For instructions, see the section.
<b>Circular reference caused by alias</b>	The alias assigned to a field is the same as a component of the expression for that field. An alias is a name that is given to any expression in the <b>Field</b> row of the query design grid that is not an actual field. Access assigns the alias for you if you do not do so yourself; for example, <b>EXPR1</b> . An alias is immediately followed by a colon (:) and then by the expression. When you run the query, the alias becomes the column name in the datasheet.	Change the alias. For instructions, see the section.
<b>#Error</b>	This error can occur when the value of a calculated field is greater than the value allowed by the field's <b>FieldSize</b> property setting. This also occurs when the denominator of a calculated field is or evaluates to zero (0).	Ensure that the calculated field's denominator does not evaluate to zero (0). If appropriate, change the <b>FieldSize</b> property.
<b>#Deleted</b>	The record being referred to has been deleted.	If the record was deleted accidentally, it must be restored from a backup. If the deletion was intentional, you can dismiss this error message by pressing SHIFT+F9 to refresh the query.

## Run a query

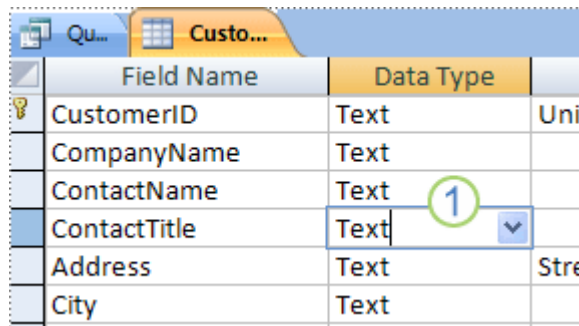
- **Check the joined fields in your query**
  - To check the data types of fields in a query, you look at the source tables in Design view and inspect the properties for the fields you are checking.
    1. Open the query in Design view. Joins appear as lines that connect fields in the source tables. Note the table and field names for each join.
    2. In the Navigation Pane, right-click each table that has one or more fields joined in your query, and then click **Design View**.



1. Joined fields with different data types.
2. Right-click the table, then click Design View.

## Run a query

- Check the joined fields in your query
  - To check the data types of fields in a query, you look at the source tables in Design view and inspect the properties for the fields you are checking.
- 3. For each join, compare the values in the **Data Type** column of the table design grid for the fields involved in that join.



Field Name	Data Type
CustomerID	Text
CompanyName	Text
ContactName	Text
ContactTitle	Text
Address	Text
City	Text

1. Check the data type of the joined fields in table Design view.

- 4. To switch to a table so that you can see its fields, click the tab with that table's name..

## Run a query

- **Compact and repair your database**

- Running the Compact and Repair Database utility within Access can improve the performance of your database.
- This utility makes a copy of the database file and, if it is fragmented, rearranges how the database file is stored on disk.
- After the compact and repair process has completed, the compacted database will have reclaimed wasted space, and is usually smaller than the original.
- By compacting the database frequently, you can help ensure optimal performance of the database application, and also resolve errors that arise from hardware problems, power failures or surges, and similar causes.
- After the compact operation has completed, query speed is enhanced because the underlying data has been rewritten to the tables in contiguous pages.
- Scanning contiguous pages is much faster than scanning fragmented pages. Queries are also optimized after each database compaction.



## Run a query

- **Compact and repair your database**

- During the compact operation, you can use the original name for the compacted database file, or you can use a different name to create a separate file.
- If you use the same name and the database is compacted successfully, Access automatically replaces the original file with the compacted version.

### Set an option that automates this process

1. Click **File > Options** to open the **Access Options** dialog box.
2. Click **Current Database** and, under **Application Options**, select the **Compact on Close** check box.

This causes Access to automatically compact and repair the database every time it is closed.

### Manually compact and repair your database

1. Click **Database Tools > Compact and Repair Database**.

## Run a query

- **Change a field alias**

1. Open the query in Design view.
2. In the query design grid, look for fields that have aliases. These will have a colon at the end of the field name, as in **Name:**.
3. Check each alias to ensure that the alias does not match the name of any field that is part of the alias' expression. If it does, change the alias.

# Use a query as the record source for a form or report

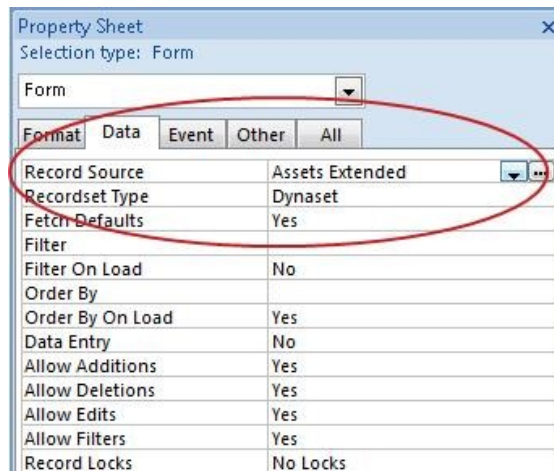
- You can use a query to supply data to a form or report in Access.
- You can use a query when you create the form or report, or you can change an existing form or report by setting its Record Source property.
- When you set the Record Source property, you can either specify an existing query, or you can create a new query to use.
- If you use a query as the record source, you might not be able to edit the data. Before you use a query as your record source, you should consider whether you need to edit data.

# Use a query as the record source for a form or report

- Use an existing query as the record source of a form or report
  - In Design view, set the **Record Source** property to an existing query that you want to use.
    1. Open the form or report in Design view.

If the property sheet is not already open, press **F4** to open it.

    2. In the property sheet, on the **Data** tab, click the **Record Source** property box.



3. Do one of the following:
  - Start typing the name of the query that you want to use.  
Access automatically fills in the name of the object as you type.
  - or
  - Click the arrow and then select the query that you want to use.

# Use a query as the record source for a form or report

- **Create a query as the record source of a form or report**
  - In Design view, use the **Build** button (Builder button) in the **Record Source** property box to create a new query to use as the record source.

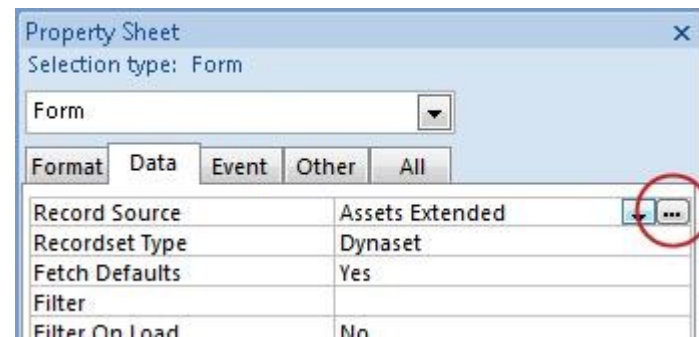
1. Open the form or report in Design view.

If the property sheet is not already open, press **F4** to open it.

2. In the property sheet, on the **Data** tab, click the **Record Source** property box.

3. Click Builder button. 

4. Design the query, and then  
save and close it.



## Examples of query criteria

- Query criteria help you zero in on specific items in an Access database. If an item matches all the criteria you enter, it appears in the query results.
- To add criteria to an Access query, open the query in Design view and identify the fields (columns) you want to specify criteria for.
- If the field is not in the design grid, double-click the field to add it to the design grid and then enter the criterion in the **Criteria** row for that field.
- A query criterion is an expression that Access compares to query field values to determine whether to include the record that contains each value. For example, = "**Chicago**" is an expression that Access can compare to values in a text field in a query. If the value for that field in a given record is "**Chicago**", Access includes the record in the query results.

## Examples of query criteria

- A criterion is similar to a formula — it is a string that may consist of field references, operators, and constants. Query criteria are also referred to as expressions in Access.
- The following tables shows some sample criteria and explains how they work.

Criteria	Description
>25 and <50	This criterion applies to a Number field, such as Price or UnitsInStock. It includes only those records where the Price or UnitsInStock field contains <b>a value greater than 25 and less than 50.</b>
DateDiff ("yyyy", [BirthDate], Date()) > 30	This criterion applies to a Date/Time field, such as BirthDate. Only records where <b>the number of years between a person's birthdate and today's date is greater than 30</b> are included in the query result.
Is Null	This criterion can be applied to any type of field to show records where <b>the field value is null.</b>



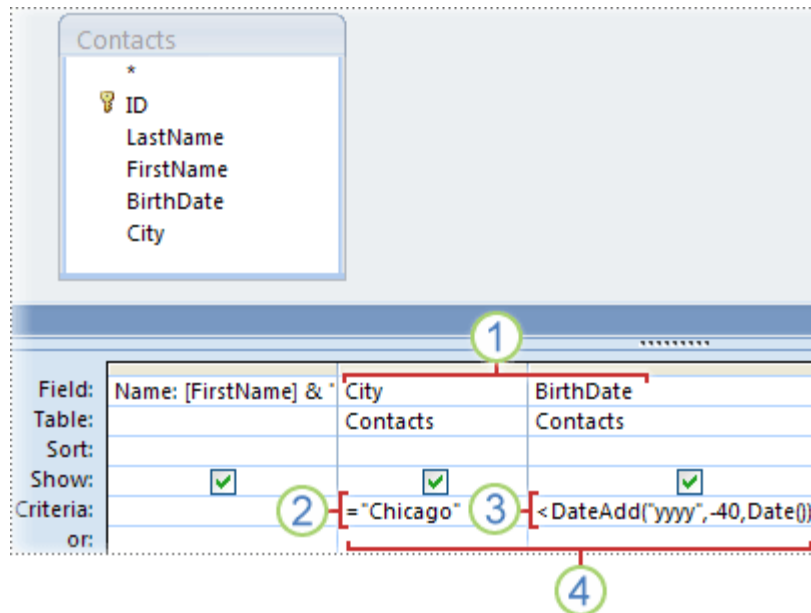
## Examples of query criteria

- As you can see, criteria can look very different from each other, depending on the data type of the field to which they apply and your specific requirements.
- Some criteria are simple, and use basic operators and constants.
- Others are complex, and use functions, special operators, and include field references.
- To add a criteria to a query, you must open the query in Design view. You then identify the fields for which you want to specify criteria. If the field is not already in the design grid, you add it by either dragging it from the query design window to the field grid, or by double-clicking the field (Double-clicking the field automatically adds it to the next empty column in the field grid.). Finally, you type the criteria in the **Criteria** row:

## Examples of query criteria

- Criteria that you specify for different fields in the **Criteria** row are combined by using the AND operator. In other words, the criteria specified in the City and BirthDate fields are interpreted like this:

**City = "Chicago" AND BirthDate < DateAdd (" yyyy ", -40, Date())**



The screenshot shows a Microsoft Access query design view. At the top, a table named 'Contacts' is listed with fields: ID (primary key), LastName, FirstName, BirthDate, and City. Below this, a query grid is shown with three columns: 'Name: [FirstName] & \*', 'City', and 'BirthDate'. The 'Table' row for all three columns is 'Contacts'. The 'Criteria' row contains the following criteria: for 'Name', '[FirstName] & \*'; for 'City', '= "Chicago"'; and for 'BirthDate', '< DateAdd("yyyy", -40, Date())'. The 'Criteria' row is highlighted with a red border. Numbered callouts (1, 2, 3, 4) are placed around the grid: 1 points to the 'City' column header, 2 points to the 'Criteria' row, 3 points to the 'BirthDate' column header, and 4 points to the 'Criteria' row.

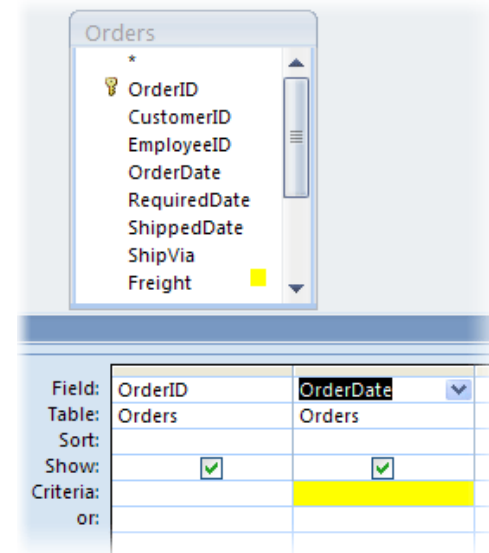
Field:	Name: [FirstName] & *	City	BirthDate
Table:	Contacts	Contacts	Contacts
Sort:			
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:		= "Chicago"	< DateAdd("yyyy", -40, Date())
or:			

1. The City and BirthDate fields include criteria.
2. Only records where the value of the City field is Chicago will satisfy this criterion.
3. Only records of those who are at least 40 years old will satisfy this criterion.
4. Only records that meet both criteria will be included in the result.



## Examples of query criteria

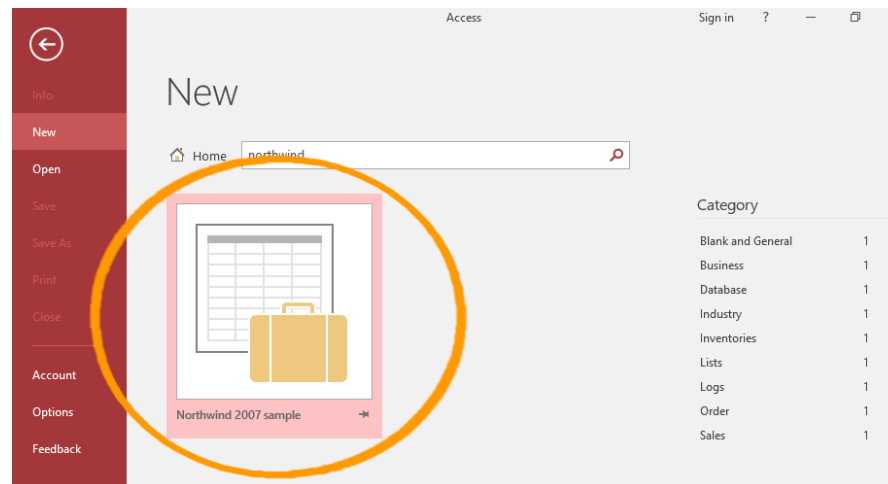
- If the criteria is temporary or changes often, you can filter the query result instead of frequently modifying the query criteria. A **filter** is a temporary criterion that changes the query result without altering the design of the query.
- If the criteria fields don't change, but the values you are interested in do change frequently, you can create a parameter query. A **parameter** query prompts the user for field values, and then uses those values to create the query criteria.
- The following examples are for the CountryRegion field in a query that is based on a table that stores contacts information. The criterion is specified in the **Criteria** row of the field in the design grid.



Field:	OrderID	OrderDate
Table:	Orders	Orders
Sort:		
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:		
or:		

# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



Using this database, please try to use different queries.



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ❑ Module 5. Practice

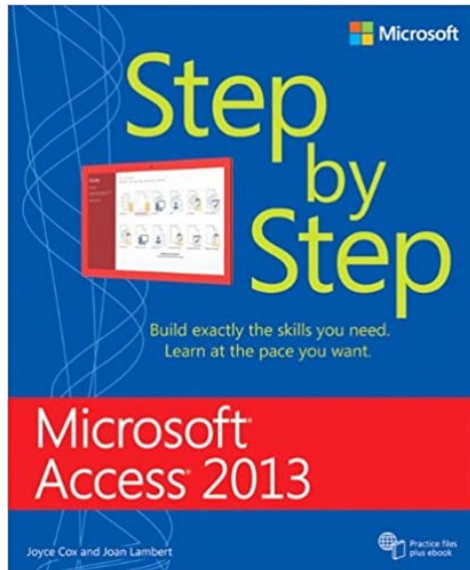
### ❑ Topic 4. Merging of data into one form. Presentation of an effective report.

#### ❑ Practical lesson 1. Merging of data into one form

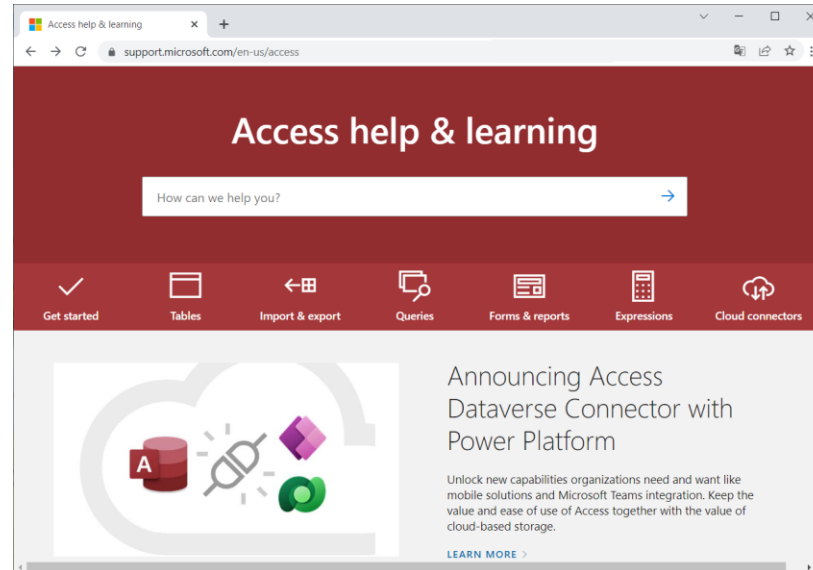




# Practical lesson. Merging of data into one form.



*Joyce Cox, Joan Lambert.  
Microsoft Access 2013 Step By  
Step, Microsoft Press., 2013.*



*<https://support.microsoft.com/en-us/access>  
<https://www.customguide.com>*

## Introduction to forms

- A form in Access is a database object that you can use to create a user interface for a database application.
- A "bound" form is one that is directly connected to a data source such as a table or query, and can be used to enter, edit, or display data from that data source.
- Alternatively, you can create an "unbound" form that does not link directly to a data source, but which still contains command buttons, labels, or other controls that you need to operate your application.

## Introduction to forms

- You can use bound forms to control access to data, such as which fields or rows of data are displayed.
- For example, certain users might need to see only several fields in a table with many fields.
- Providing those users with a form that contains only those fields makes it easier for them to use the database.
- You can also add command buttons and other features to a form to automate frequently performed actions.

## Introduction to forms

- Think of bound forms as windows through which people see and reach your database.
- An effective form speeds the use of your database, because people don't have to search for what they need.
- A visually attractive form makes working with the database more pleasant and more efficient, and it can also help prevent incorrect data from being entered.

## Create a form by using the Form tool

- You can use the Form tool to create a form with a single mouse-click.
- When you use this tool, all the fields from the underlying data source are placed on the form.
- You can start using the new form immediately, or you can modify it in Layout view or Design view to better suit your needs.
- Use the Form tool to create a new form:

## Create a form by using the Form tool

- Use the Form tool to create a new form:
  1. In the Navigation Pane, click the table or query that contains the data you want to see on your form.
  2. On the **Create** tab, in the **Forms** group, click **Form**.
- Access creates the form and displays it in Layout view. In Layout view, you can make design changes to the form while it is displaying data.
- For example, you can adjust the size of the text boxes to fit the data, if necessary.

## Create a form by using the Form tool

- If Access finds a single table that has a one-to-many relationship with the table or query that you used to create the form, Access adds a datasheet to the form that is based on the related table or query.
- For example, if you create a simple form that is based on the Employees table, and there is a one-to-many relationship that is defined between the Employees table and Orders table, the datasheet displays all the records in the Orders table that relate to the current Employee record.



## Create a form by using the Form tool

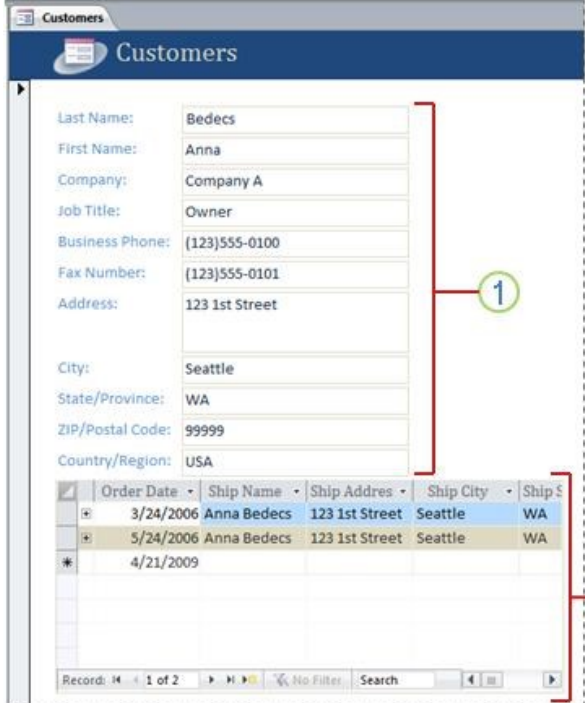
- You can delete the datasheet from the form if you decide you do not need it.
- If there is more than one table with a one-to-many relationship to the table that you used to create the form, Access does not add any datasheets to the form.

## Create a form by using the Form tool

- You can use the Form tool in Access to quickly create a single item form. This type of form displays information about one record at a time, as shown in the following illustration:

1. The form displays information for a single record.
2. In some cases, Access adds a subdatasheet to display related information.

When you use the Form tool, all the fields from the underlying data source are added to the form. You can start to use the new form immediately, or you can modify it in Layout view or Design view to better suit your needs.



The screenshot shows an Access form titled "Customers" in Layout View. The form displays a single record for a customer named Anna Bedecs, with fields for Last Name, First Name, Company, Job Title, Business Phone, Fax Number, Address, City, State/Province, ZIP/Postal Code, and Country/Region. A red bracket labeled "1" groups these fields. Below the main form is a subdatasheet titled "Orders" with columns for Order Date, Ship Name, Ship Address, Ship City, and Ship State. The subdatasheet contains three records, with the first two highlighted. A red bracket labeled "2" groups the subdatasheet. The status bar at the bottom indicates "Record: 1 of 2".

ERASMUS+

## Create a split form by using the Split Form tool

- A split form gives you two views of the data at the same time — a Form view and a Datasheet view.
- A **split** form differs from a form/subform combination in that the two views are connected to the same data source and are synchronized with one another at all times.
- Selecting a field in one part of the form selects the same field in the other part of the form.

## Create a split form by using the Split Form tool

- You can add, edit, or delete data from either part (as long as the record source is updatable, and you have not configured the form to prevent these actions).
- Working with split forms gives you the benefits of both kinds of forms in a single form.
- For example, you can use the datasheet portion of the form to quickly locate a record, and then use the form portion to view or edit the record.

## Create a split form by using the Split Form tool

- To create a split form by using the Split Form tool:
  1. In the Navigation Pane, click the table or query that contains the data that you want on your form. Or open the table or query in Datasheet view.
  2. On the **Create** tab, in the **Forms** group, click **More Forms**, and then click **Split Form**.
- Access creates the form and displays it in Layout view. In Layout view, you can make design changes to the form while it is displaying data. For example, you can adjust the size of the text boxes to fit the data, if necessary.

# Create a form that displays multiple records by using the Multiple Items tool

- When you create a form by using the Form tool, the form that Access creates displays a single record at a time. If you want a form that displays **multiple** records but is more customizable than a datasheet, you can use the Multiple Items tool.
  1. In the Navigation Pane, click the table or query that contains the data you want to see on your form.
  2. On the **Create** tab, in the **Forms** group, click **More Forms**, and then click **Multiple Items**.

## Create a form that displays multiple records by using the Multiple Items tool

- Access creates the form and displays it in Layout view. In Layout view, you can make design changes to the form while it is displaying data.
- When you use the Multiple Items tool, the form that Access creates resembles a datasheet. The data is arranged in rows and columns, and you see more than one record at a time.
- However, a Multiple Items form gives you more customization options than a datasheet, such as the ability to add graphical elements, buttons, and other controls.



# Create a form that displays multiple records by using the Multiple Items tool

- A multiple item form, also known as a continuous form, lets you show information from more than one record at a time.
- The data is arranged in rows and columns (similar to a datasheet), and multiple records are displayed at a time.
- However, because it is a form, there are more customization options than with a datasheet. You can add features such as graphical elements, buttons, and other controls.

# Create a form that displays multiple records by using the Multiple Items tool

- A multiple item form can resemble a datasheet when you first create it, as shown in the following illustration:

The screenshot shows a software interface for a 'Customers' form. At the top, there is a header bar with the title 'Customers' and a small icon of a document with a list. Below the header, there is a table with five columns: 'ID', 'Company', 'Last Name', 'First Name', and 'Job Title'. The table contains ten rows of data, numbered 1 through 10. The first row shows 'Company A' with 'Bedecs' as the last name, 'Anna' as the first name, and 'Owner' as the job title. The subsequent rows follow a similar pattern with different company and person names. The table is enclosed in a dashed border, suggesting it is a preview or a mockup of a form.

ID	Company	Last Name	First Name	Job Title
1	Company A	Bedecs	Anna	Owner
2	Company B	Gratacos Solsona	Antonio	Owner
3	Company C	Axen	Thomas	Purchasing Rep
4	Company D	Lee	Christina	Purchasing Mar
5	Company E	O'Donnell	Martin	Owner
6	Company F	Pérez-Olaeta	Francisco	Purchasing Man
7	Company G	Xie	Ming-Yang	Owner
8	Company H	Andersen	Elizabeth	Purchasing Rep
9	Company I	Mortensen	Sven	Purchasing Mar
10	Company J	Wacker	Roland	Purchasing Mar

# Create a form by using the Form Wizard

- To be more selective about what fields appear on your form, you can use the **Form Wizard** instead of the various form-building tools previously mentioned.
- You can also define how the data is grouped and sorted, and you can use fields from more than one table or query, as long as you specified the relationships between the tables and queries beforehand.

# Create a form by using the Form Wizard

1. On the **Create** tab, in the **Forms** group, click **Form Wizard**.

2. Follow the directions on the pages of the Form Wizard.

**Note:** *If you want to include fields from multiple tables and queries on your form, do not click **Next** or **Finish** after you select the fields from the first table or query on the first page of the Form Wizard. Instead, repeat the steps to select a table or query, and click any additional fields that you want to include on the form. Then click **Next** or **Finish** to continue.*

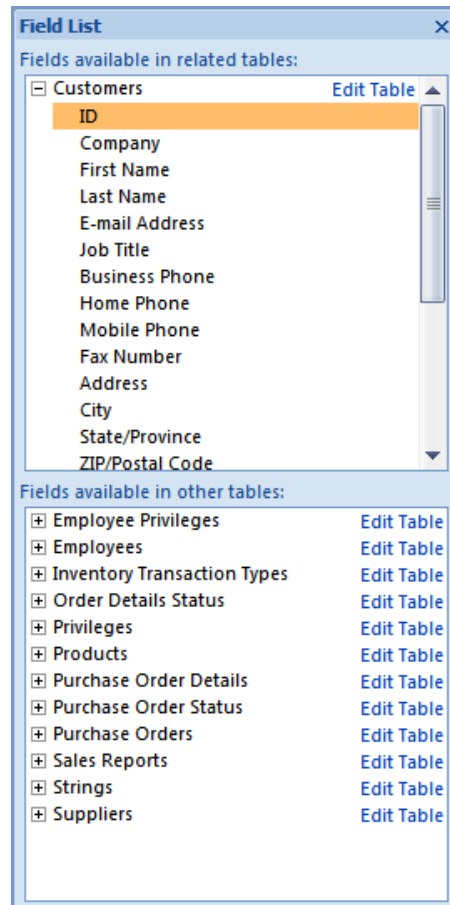
3. On the last page of the wizard, click **Finish**.

## Create a form by using the Blank Form tool

- If the wizard or the form-building tools don't meet your needs, you can use the **Blank Form** tool to build a form. This can be a very quick way to build a form, especially if you plan to put only a few fields on your form.
  1. On the **Create** tab, in the **Forms** group, click **Blank Form**.

Access opens a blank form in Layout view, and displays the **Field List** pane.
  2. In the **Field List** pane, click the plus sign (+) next to the table or tables that contain the fields that you want to see on the form.

# Create a form by using the Blank Form tool



The order of the tables in the Field List pane can change, depending on which part of the form is currently selected. If you are not able to add a field to the form, try selecting a different part of the form and then try adding the field again.

## Create a form by using the Blank Form tool

3. To add a field to the form, double-click it or drag it onto the form.
  - After the first field has been added, you can add several fields at once by holding down the CTRL key, clicking several fields, and then dragging them onto the form at the same time.
  - The order of the tables in the **Field List** pane can change, depending on which part of the form is currently selected. If the field you want to add is not visible, try selecting a different part of the form and then try adding the field again.
4. Use the tools in the **Header/Footer** group on the **Design** tab to add a logo, title, or the date and time to the form.
5. Use the tools in the **Controls** group of the **Design** tab to add a wider variety of controls to the form.

For a slightly larger selection of controls, switch to Design view by right-clicking the form and then clicking **Design View**.



# Understand Layout view and Design view

- **Layout view** Layout view is the most intuitive view to use for form modification, and it can be used for almost all the changes that you would want to make to a form in Access.
- In Layout view, the form is actually running. Therefore, you can see your data much as it will appear when you are using the form. However, you can also change the form design in this view. Because you can see the data while you are modifying the form, this is a very useful view for setting the size of controls or performing almost any other task that affects the appearance and usability of the form.
- If you encounter a task that cannot be performed in Layout view, you can switch to Design view. In certain situations, Access displays a message that states that you must switch to Design view before you can make a particular change.

# Understand Layout view and Design view

- **Design view** Design view gives you a more detailed view of the structure of your form. You can see the Header, Detail, and Footer sections for the form. The form is not actually running when it is shown in Design view. Therefore, you cannot see the underlying data while you are making design changes. However, there are certain tasks that you can perform more easily in Design view than in Layout view. You can:
  - Add a wider variety of controls to your form, such as bound object frames, page breaks, and charts.
  - Edit text box control sources in the text boxes themselves, without using the property sheet.
  - Resize form sections, such as the Form Header or the Detail section.
  - Change certain form properties that cannot be changed in Layout view.

## Fine-tune your form in Layout view

- After you create a form, you can easily fine-tune its design by working in Layout view. Using the actual form data as your guide, you can rearrange the controls and adjust their sizes. You can place new controls on the form and set the properties for the form and its controls.
- To switch to Layout view, right-click the form name in the Navigation Pane and then click **Layout View**.
- Access shows the form in Layout view.
- You can use the property sheet to change the properties for the form and its controls and sections. To display the property sheet, press F4.

## Fine-tune your form in Layout view

- You can use the **Field List** pane to add fields from the underlying table or query to your form design. To display the **Field List** pane:
  - On the **Design** tab, in the **Tools** group, click **Add Existing Fields** or use the keyboard shortcut by pressing ALT+F8.
- You can then drag fields directly from the **Field List** pane onto your form.
  - To add a single field, double-click it or drag it from the **Field List** pane to the section on the form where you want it displayed.
  - To add several fields at once, hold down CTRL and click the fields that you want to add. Then drag the selected fields onto the form.

## Fine-tune your form in Design view

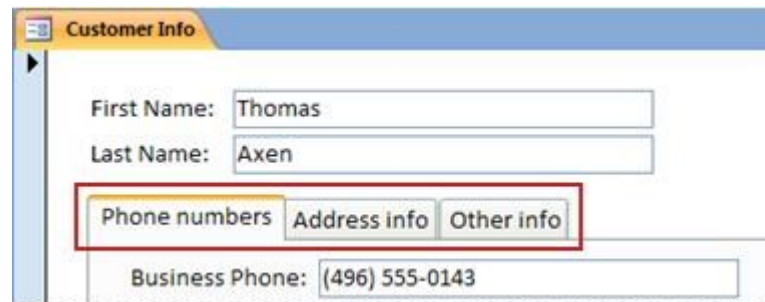
- You can also fine-tune your form's design by working in Design view. You can add new controls and fields to the form by adding them to the design grid. The property sheet gives you access to many properties that you can set to customize your form.
- To switch to Design view, right-click the form name in the Navigation Pane and then click **Design View**.
- Access shows the form in Design view.
- You can use the property sheet to change the properties for the form and its controls and sections. To display the property sheet, press F4.

## Fine-tune your form in Design view

- You can use the **Field List** pane to add fields from the underlying table or query to your form design. To display the **Field List** pane:
  - On the **Design** tab, in the **Tools** group, click **Add Existing Fields** or use the keyboard shortcut by pressing ALT+F8.
- You can then drag fields directly from the **Field List** pane onto your form.
  - To add a single field, double-click it or drag it from the **Field List** pane to the section on the form where you want it displayed.
  - To add several fields at once, hold down CTRL and click the fields that you want to add. Then drag the selected fields onto the form.

## Create a tabbed form

- Adding tabs to an Access form can make it more organized and easier to use, especially if the form contains many controls. By placing related controls on separate pages of the tab control, you can reduce clutter and make it easier to work with your data.

A screenshot of a Microsoft Access form titled 'Customer Info'. The form has a blue header bar with the title. Below the header, there are two text boxes: 'First Name: Thomas' and 'Last Name: Axen'. Below these, there is a tabbed control with three tabs: 'Phone numbers', 'Address info', and 'Other info'. The 'Phone numbers' tab is currently selected and highlighted with a red border. Below the tabs, there is a text box labeled 'Business Phone: (496) 555-0143'.

- To add tabs to a form, you use the Tab Control tool. Each page of a tab control acts as a container for other controls, such as text boxes, combo boxes, or command buttons.



## Create a tabbed form

- **Add a tab control to a form**
  1. On the **Design** tab, in the **Controls** group, click the **Tab Control** tool.
  2. Click on the form where you want to place the tab control.

Access places the tab control on the form.

## Create a tabbed form

- **Move existing controls to a tab page**
  1. Select the controls that you want to move to the tab page. To select multiple controls, hold down the SHIFT key and then click the controls you want to move.
  2. On the **Home** tab, in the **Clipboard** group, click **Cut**.
  3. Click the label text on the tab that corresponds to the page on which you want to place the controls. A selection box will appear on the tab page.
  4. On the **Home** tab, in the **Clipboard** group, click **Paste**.

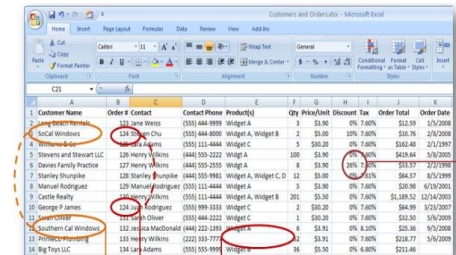
## Create a tabbed form

- **Drag fields from the Field List task pane to a tab page**
  1. Select the tab page to which you want to add the fields.
  2. On the **Design** tab, in the **Tools** group, click **Add Existing Field**.
  3. Navigate to the table that contains the fields you want to add.
  4. Drag each field from the Field List task pane to the tab page. Alternatively, select several fields by holding down the CTRL or SHIFT key while clicking the fields, and then drag them all to the tab page at the same time.
  5. Release the mouse button.

# Merge Tables In Access Database:

## 3 Ways To Merge Tables In Access Database

- Occasionally, copying and pasting content may not be the most efficient method for transferring data, especially when dealing with large amounts of data spread across multiple tables in an Access database.
- To simplify this process, Access provides the option to merge database tables, allowing you to maintain an organized database structure effortlessly.
- There are three approaches offered by Access database to merge tables or consolidate rows/columns in primary tables. This lesson will provide you with detailed, step-by-step instructions on how to perform each of these methods for merging Access database tables.



Customer Name	Order #	Contact	Phone	Product(s)	Qty	Price/Unit	Discount	Tax	Order Total	Order Date
SoCal Windows	124	John Weiss	(555) 444-8999	Widget A	3	\$1.90	0%	7.68%	\$12.39	1/2/2008
SoCal Windows	125	John Weiss	(555) 444-8999	Widget A, Widget B	2	\$5.00	20%	7.68%	\$10.76	2/8/2008
SoCal Windows	126	John Weiss	(555) 444-8999	Widget C	5	\$30.20	0%	7.68%	\$162.40	2/1/2007
Stevens and Stewart LLC	126	Henry Williams	(444) 555-2222	Widget A	100	\$1.90	0%	7.68%	\$190.00	5/8/2005
Stevens Family Practice	127	Henry Williams	(444) 555-2222	Widget A	5	\$1.90	0%	7.68%	\$10.39	7/27/1999
Stanley Shurgate	128	Stanley Shurgate	(444) 555-9901	Widget A, Widget C, D	12	\$1.90	0%	7.68%	\$22.80	8/5/1999
Manuel Rodriguez	129	Manuel Rodriguez	(555) 121-4444	Widget A	5	\$1.90	0%	7.68%	\$10.39	6/19/2001
Castle Realty	130	Castle Realty	(555) 121-4444	Widget A, Widget B	200	\$5.00	0%	7.68%	\$1,039.52	12/14/2001
George P James	131	George P James	(555) 999-3333	Widget C	2	\$30.20	0%	7.68%	\$64.99	3/23/2007
SoCal Windows	132	John Weiss	(555) 444-2222	Widget C	1	\$40.20	0%	7.68%	\$142.30	5/6/2009
Customer Car Windows	132	John Weiss	(555) 444-2222	Widget C	1	\$40.20	0%	7.68%	\$142.30	5/6/2009
Customer Car Windows	133	Henry Williams	(222) 333-7777	Widget C	1	\$40.20	0%	7.68%	\$142.30	5/6/2009
Big Toys LLC	134	Lane Adams	(555) 555-8999	Widget C	36	\$5.50	0%	6.85%	\$211.40	

Lack of controls can cost you money. Excel provides only limited validation capabilities to prevent data entry errors. In this case, is a 26% discount allowed for this product?

Missing Data and Inconsistencies make it hard to create reliable reports

Bad data can mean you don't see the full picture. If you look to see how much the 'SoCal Windows' account is worth, you won't get the full picture.

Duplicate values can lead to mistakes that cost you time and money, like an accidentally duplicated order number. When you need to trace order 124, which customer did it go to?

## Why The Need Of Merging Data In Access?

- Merging data in Access enables you to conveniently consolidate and analyze vast amounts of information. Let's say you receive inputs from various sources, such as all members of your account executive team. Creating a report from a single file is straightforward and can be accomplished by employing the simple technique of copy and paste.
- However, when dealing with multiple records and tables, copying and pasting data from numerous rows and columns becomes inadequate.
- Hence, merging Access tables is the ideal solution for such tasks.

	A	B	C	D	E	F
1	ID	Company	Last Name	First Name	E-mail Address	Job Title
2	1	Company A	Bedecs	Anna		Owner
3	2	Company B	Gratacos Solsona	Antonio		Owner
4	3	Company C	Axen	Thomas		Purchasing Represent
5	4	Company D	Lee	Christina		Purchasing Manager
6	5	Company E	O'Donnell	Martin		Owner
7	6	Company F	Pérez-Olaeta	Francisco		Purchasing Manager
8	7	Company G	Xie	Ming-Yang		Owner
9	8	Company H	Andersen	Elizabeth		Purchasing Represent
10	9	Company I	Mortensen	Sven		Purchasing Manager
11	10	Company J	Wacker	Roland		Purchasing Manager
12	11	Company K	Krschne	Peter		Purchasing Manager
13	12	Company L	Edwards	John		Purchasing Manager
14	13	Company M	Ludick	Andre		Purchasing Represent
15	14	Company N	Grilo	Carlos		Purchasing Represent
16	15	Company O	Kupkova	Helena		Purchasing Manager
17	16	Company P	Goldschmidt	Daniel		Purchasing Represent

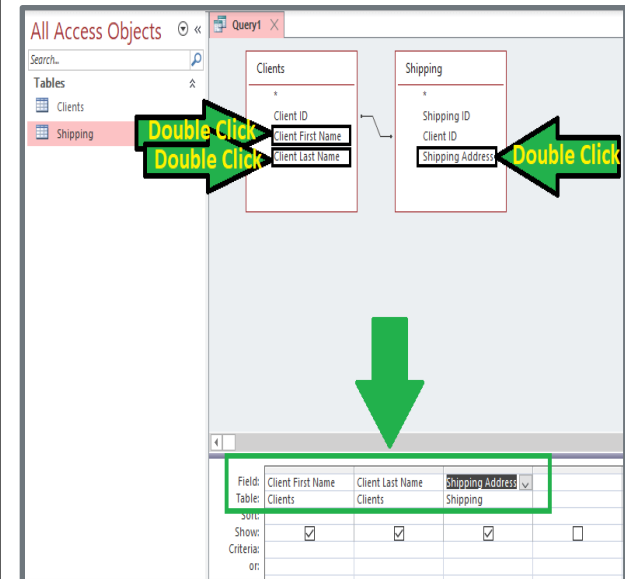
# How To Merge Tables In Access Database?

## Method 1 : Merge Access Tables Using Append Method

As we are familiar, the Access database primarily comprises tables. If your database contains two tables that share similarities, there is a convenient way to merge them without going through a laborious manual process. The solution lies in utilizing the append query.

By employing this specific append query command, you can effortlessly merge table data in Access.

Furthermore, you have the flexibility to choose the desired records from the source table and append them to the destination table.



## Step 1

- To begin the process of merging tables in Access, open the Access database that contains the table you want to merge with another table.
- It is important to ensure that the data types of the source and destination tables are compatible with each other.
- For example, if the first field of your source table is of the number data type, the corresponding field in the destination table must also be of the number data type.
- According to Microsoft, text fields are generally compatible with other field types. So, if your source table has a number field and the destination table has a text field, there shouldn't be any issues during the merging process.

Clients X		
Client ID	Client First Name	Client Last Name
111	Bob	Lee
222	James	Ford
333	Nancy	Silva
444	Maria	Green
555	Jack	May



## Step 2

- Click on the "Create" button and then select "Query Design." This action will open the "Show Table" window, where you will find a list of all the tables in the database.
- Choose the table that contains the records you want to copy.
- Click on "Add" and then select "Close." Access database will add the fields and table to the query designer.

## Step 3

- In the query design, you will notice an asterisk located at the top section. Double-click on it, and Access will automatically include all the fields from the tables in a grid within the query design.



## Step 4

- Click on the "Run" button to execute the query and view the table that contains the desired records.
- Navigate to View > Design View > Append option. This action will open the Append window.

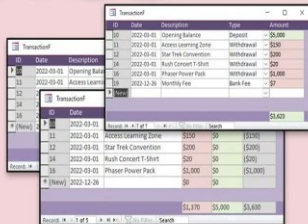

## Step 5

- If your destination table is within the current database, select "Current Database".
- Then, click on the combo box for "Table Name" and choose the table where you want to merge the records from the source table.
- Alternatively, if the destination table is located in another Access database, select the "Another Database" option. Provide the name and location of the Access database that contains the destination table.
- Enter the name of the table in the "Table Name" field and click the OK button.

# Transaction Tables

Tech Help  
599cd.com / TechHelp

Three Methods for  
Creating Tables to Track  
Transactions in  
Microsoft Access

**Access Learning Zone**

**Richard Rost**  
Microsoft MVP  
Microsoft 2014, 2015 MVP

## Step 6

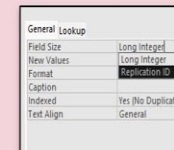
- Right-click on the query design window and select "Datasheet View" to preview the records that will be appended by the query.
- For the top section of the window, right-click and choose the "Design View" option.

## Step 7

- Click on the View tab and then select the Data Sheet > Run option.
- In the dialog box that appears, you will be prompted to choose whether to append rows from the source table to the destination table in the Access database.
- Click on the "Yes" button to proceed with the merging of tables in Access.

# Replication ID

What is a Replication ID in  
**Microsoft Access** and When  
Would You Ever Use One?



(91D8C656-1709-4D6D-ADAE-FA3298FEC98B)  
(BD125900-0064-4D2C-8DF3-A211F1FA25CB)  
(7900F156-E728-4988-B986-4709F7857CCA)  
(5FB8E508-7DF9-4546-8561-F31945C9FD81)  
(10C690-E939-4718-8828-094D34144FDA)  
(5AE004-1F6B-4999-8F70-C47C5F0F75C5)  
(49BA-5D5C-442D-AD10-FFF7D63BCCFE)  
(0315C-B039-4BF6-86AA-C531D84DEF06)  
(564C1-4DD0-49C1-AB10-8DD1FFD99306)  
(041A-AF6D-42AB-AEF1-6358E7A9DEA2)  
(A1F692-DD91-41BF-81F1-B632A67F845)  
(9D70B-13D1-4F72-AB2C-AC38E1D34771)  
(20679DEC-7989-45BA-B65E-B831844EF186)  
(FAC61E9-3E4C-42EE-945D-999F892D5718)  
(FD8E270-8475-4A10-84DF-BD2B4D7C34E5)  
(A307F556-07E6-47E1-8B48-146525EB6534)

*Fast Tips*



**Richard Rost**  
2023 Microsoft MVP



## Method 2 : Merge Access Tables Using Inner Join

- Another approach to merge tables in Access is by utilizing the inner join method.
- Here are the step-by-step instructions to perform the merging process of Access database tables using an inner join.
- Typically, an inner join is employed to retrieve complete records from linked tables. It allows you to identify common values present in both Access database tables.

# Union Query

Combine the records from multiple tables or queries into a single dataset with a UNION query in Microsoft Access

Customers			
CustomerID	FirstName	LastName	Phone
1	Richard	Rost	7167917510
2	James	Kirk	5551112222
3	Deanna	Troiwdwdw	2391118888
4	Jean Luc	Picard	+33670303532
5	Willy	Riker	
6	Malcom	Reynolds	

Employees			
EmployeeID	FirstName	LastName	Phone
1	Richard	Emp	1231231231
2	Zoe	Boots	2233223322
	(None)		

UnionQ			
ID	FirstName	LastName	Phone
2	Zoe	Boots	2233223322
1	Zebrad	Emp	1231231231
2	James	Kirk	5551112222
4	Jean Luc	Picard	+33670303532
6	Malcom	Reynolds	
5	Willy	Riker	
1	Richard	Rost	7167917510
3	Deanna	Troiwdwdw	2391118888

Tech Help  
599cd.com / TechHelp



Richard Rost  
Microsoft 2014, 2015 MVP



## Steps To Merge Tables In Access By Using The Inner Join Function

### Step 1: Create the Tables

To begin, you must create a table that you wish to link in your Access database.

As an example, I have created two tables: one for shipping information and another for client details.

Here is a screenshot displaying the client and shipping tables.

Finally, you will need to establish a link between the shipping table and the client table using the Client ID field, which should be present in both tables.

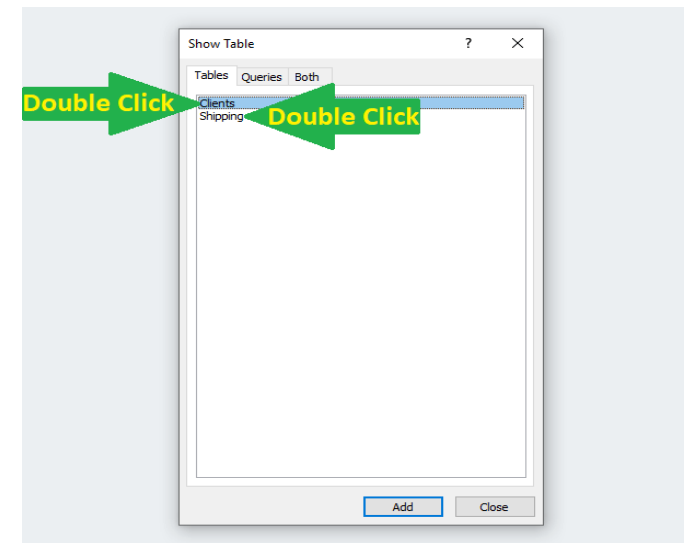
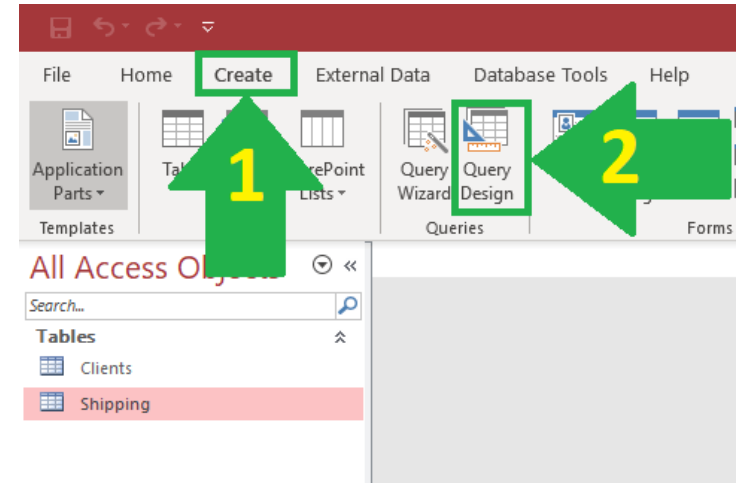
Clients		
Client ID	Client First Name	Client Last Name
111	Bob	Lee
222	James	Ford
333	Nancy	Silva
444	Maria	Green
555	Jack	May

Shipping		
Shipping ID	Client ID	Shipping Address
1235	111	21-Paris street France
1236	222	32-Rome street Italy
1237	333	22-Tokyo street Japan
1238	444	15-Rio street Brazil



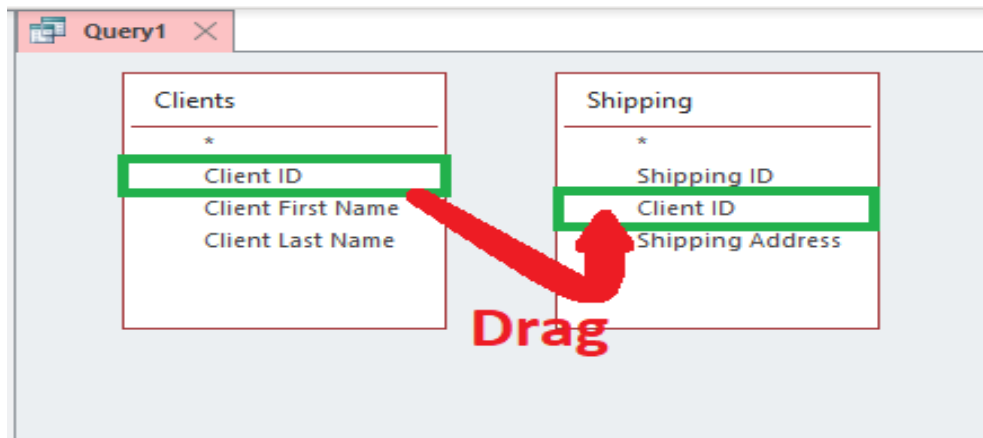
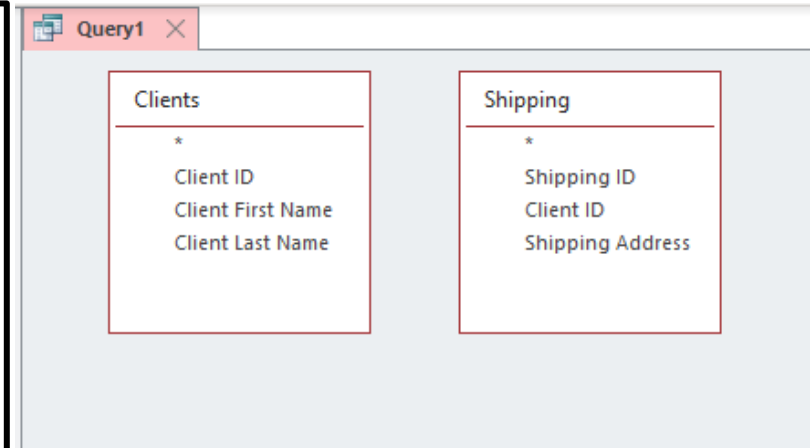
## Step 2: Link the Tables

- To link tables in Access database, begin by navigating to the Create tab and selecting Query Design.
- In the "Show Table" dialog box, double-click on each client and shipping table to include them.
- Once done, click on the Close option.



## Step 2: Link the Tables

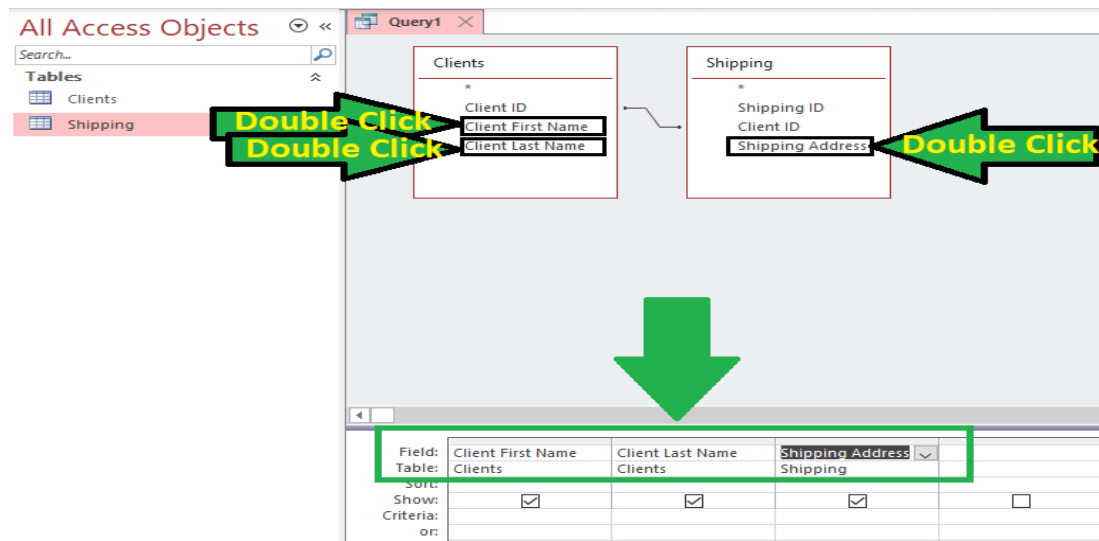
- From the client table, drag the 'Client ID' field and place it over the shipping table.
- This will establish a link between the shipping and client tables using the common field 'Client ID'.





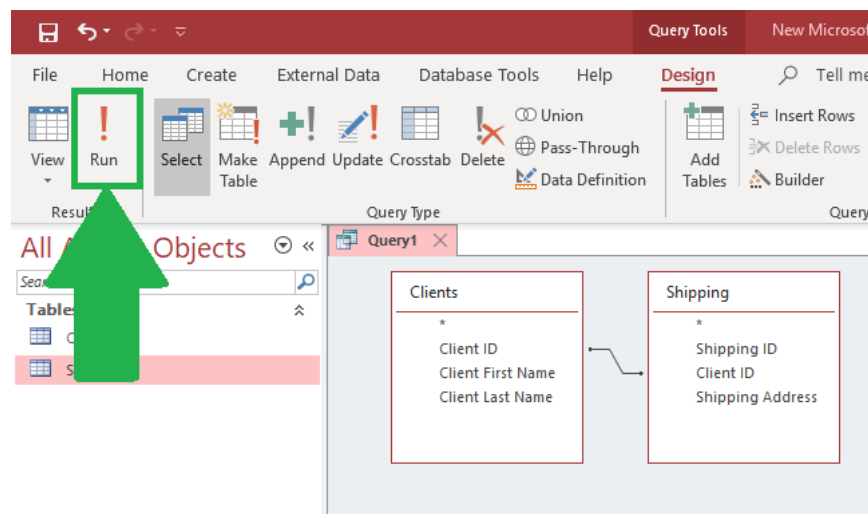
### Step 3: Select The Fields To Display

- Select the fields you want to display from the linked tables. To do this, double-click on the desired fields in each table.
- For example, double-click on the following fields:
- From the Clients table, choose 'Client First Name' and 'Client Last Name'. From the Shipping table, choose 'Shipping Address'.



### Step 3: Select The Fields To Display

- At last hit the run button, as this will display the result:



- The newly created merged table consists of the three fields that we selected from the linked tables.
- Subsequently, you will observe that these three fields appear in the linked tables. This is the process of merging Access table data using the inner join method.

Client First Name	Client Last Name	Shipping Address
Bob	Lee	21-Paris street France
James	Ford	32-Rome street Italy
Nancy	Silva	22-Tokyo street Japan
Maria	Green	15-Rio street Brazil

ERASMUS+

### **Method 3 : Merge Access Table Using Left Joins**

- To retrieve the data, you can utilize either right or left joins. Here, I will demonstrate an example focusing on achieving the query using left joins.
- Let's consider the following scenario:

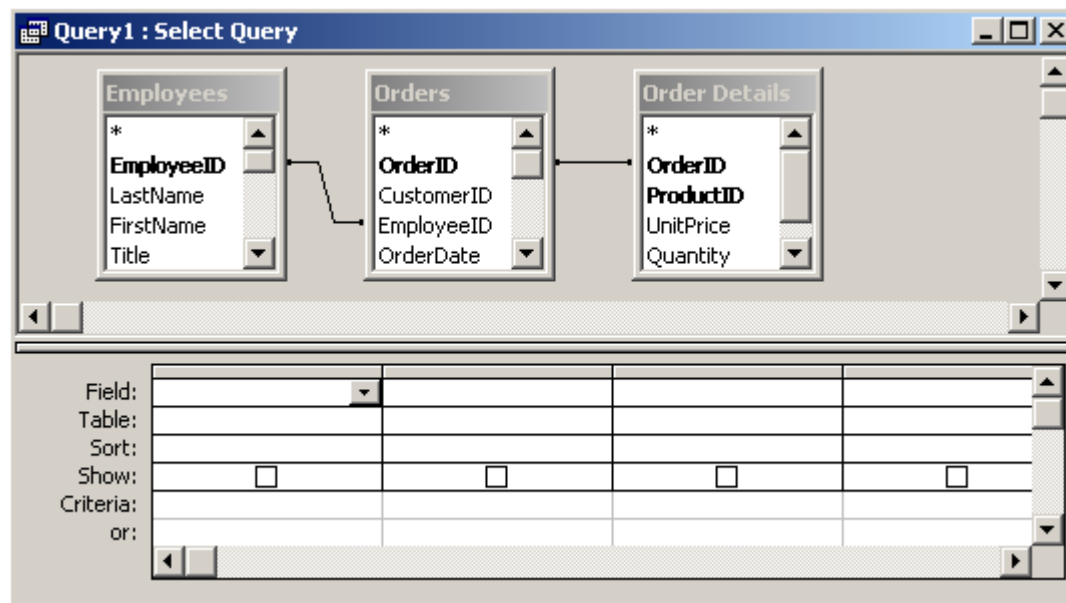
The first table is named "Employees.

" The second table is named "Orders.

" The third table is named "Order Details."

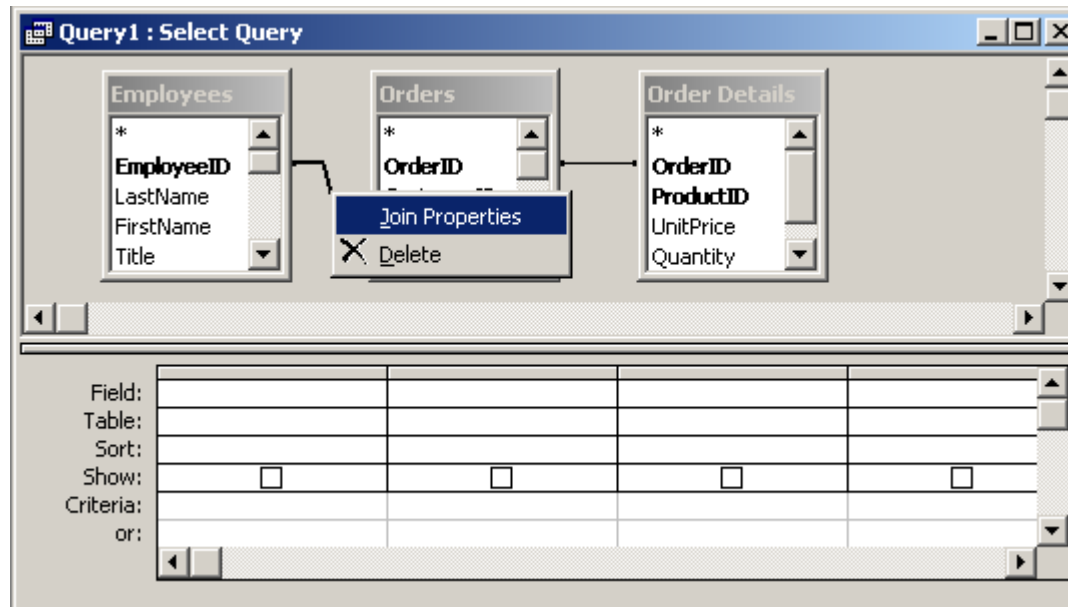
### Method 3 : Merge Access Table Using Left Joins

- To begin, create a new query and add all three tables into the query.



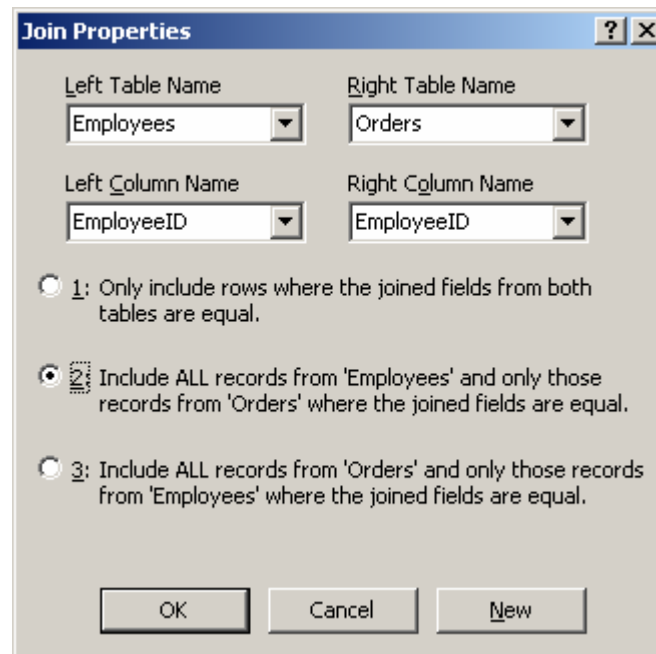
### **Method 3 : Merge Access Table Using Left Joins**

- Right-click on the join line between the "Orders" and "Employees" tables. From the popup menu, select "Join Properties."



### **Method 3 : Merge Access Table Using Left Joins**

- In the "Join Properties" window that appears, select the second option and click the OK button.



**Join Properties**

Left Table Name: Employees  
Right Table Name: Orders

Left Column Name: EmployeeID  
Right Column Name: EmployeeID

☐ 1: Only include rows where the joined fields from both tables are equal.

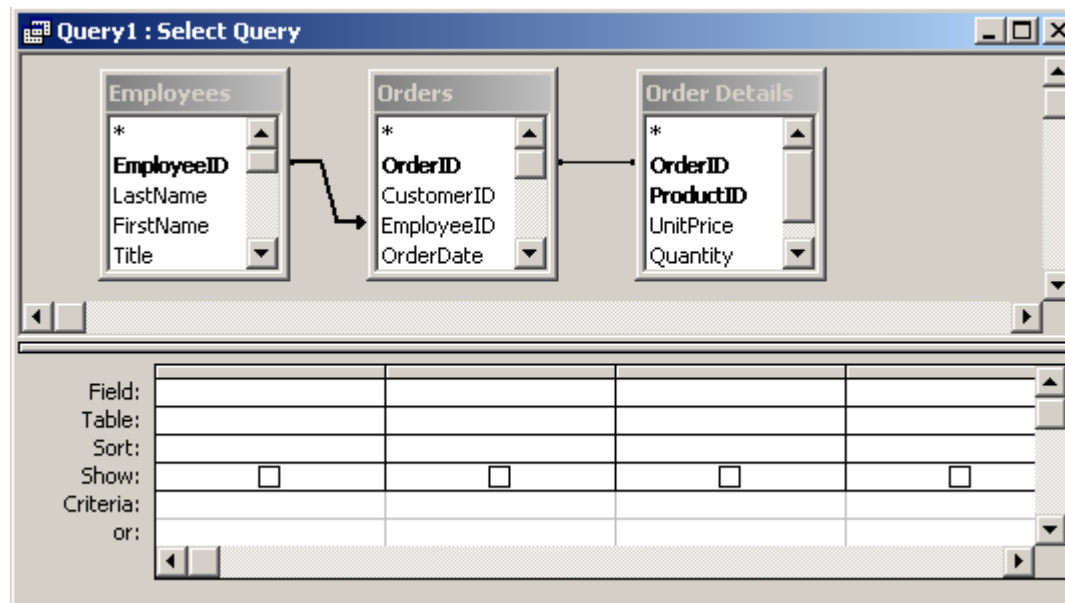
☒ 2: Include ALL records from 'Employees' and only those records from 'Orders' where the joined fields are equal.

☐ 3: Include ALL records from 'Orders' and only those records from 'Employees' where the joined fields are equal.

OK Cancel New

## Method 3 : Merge Access Table Using Left Joins

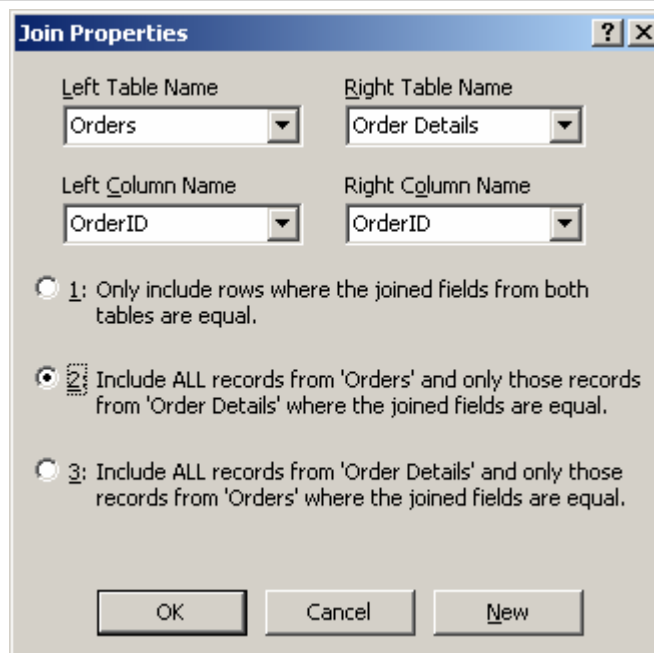
- This will modify your query appearance, displaying an arrow symbol on the right side of the join line connecting the Orders and Employees tables.





### **Method 3 : Merge Access Table Using Left Joins**

- Next, right-click on the join line between the Order Details and Orders tables.  
From the popup menu, select the "Join Properties" option.
- In the opened Properties window, choose the second option and click the OK button.



**Join Properties**

Left Table Name: Orders  
Right Table Name: Order Details

Left Column Name: OrderID  
Right Column Name: OrderID

☐ 1: Only include rows where the joined fields from both tables are equal.

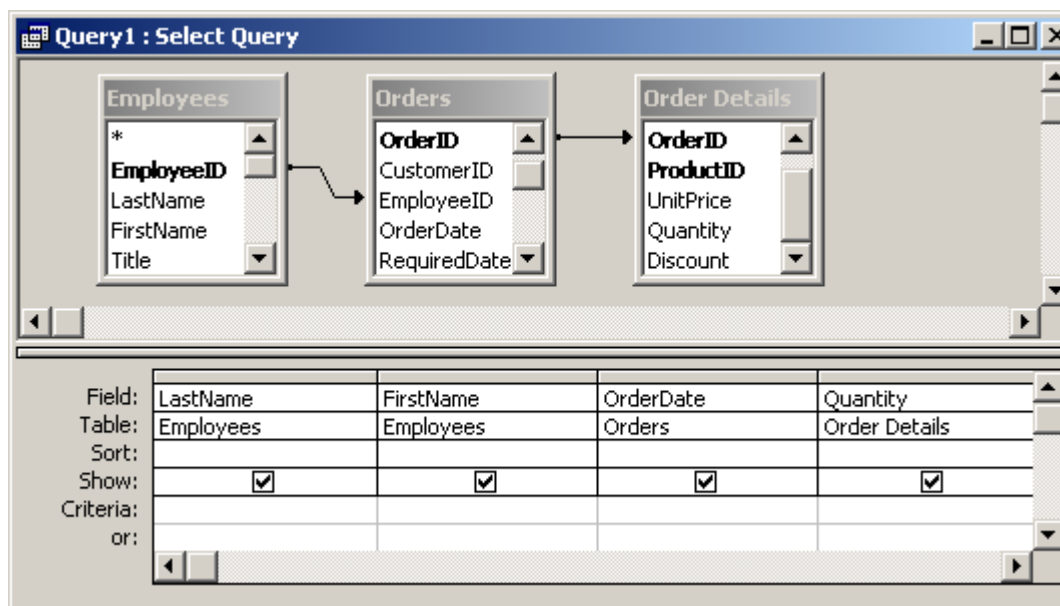
☒ 2: Include ALL records from 'Orders' and only those records from 'Order Details' where the joined fields are equal.

☐ 3: Include ALL records from 'Order Details' and only those records from 'Orders' where the joined fields are equal.

OK Cancel New

### Method 3 : Merge Access Table Using Left Joins

- Select the fields that you want to include in the query results. I have chosen the following fields:



### Method 3 : Merge Access Table Using Left Joins

- When the query is executed, you will notice that the Quantity fields and Order Date may contain blank values.

Query1 : Select Query

	Last Name	First Name	Order Date	Quantity
▶	Fuller	Andrew		
	Leverling	Janet	02-May-95	60
	Leverling	Janet	04-Sep-95	40
	Leverling	Janet	07-Apr-95	
	Leverling	Janet	19-Apr-95	30
	Leverling	Janet	19-Apr-95	28
	Leverling	Janet	19-Apr-95	60
	Leverling	Janet	19-Apr-95	30
	Leverling	Janet	24-Apr-95	14
	Leverling	Janet	04-May-95	5
	Leverling	Janet	24-Apr-95	3
	Leverling	Janet	08-Sep-95	5
	Leverling	Janet	02-May-95	20
	Leverling	Janet	09-Aug-95	24
	Leverling	Janet	09-Aug-95	30
	Leverling	Janet	08-Aug-95	25

Record: 1 of 321

## Method 3 : Merge Access Table Using Left Joins

- When the query is executed, you will notice that the Quantity fields and Order Date may contain blank values.

This occurs because there are no matching records available in the respective tables that satisfy the join criteria.

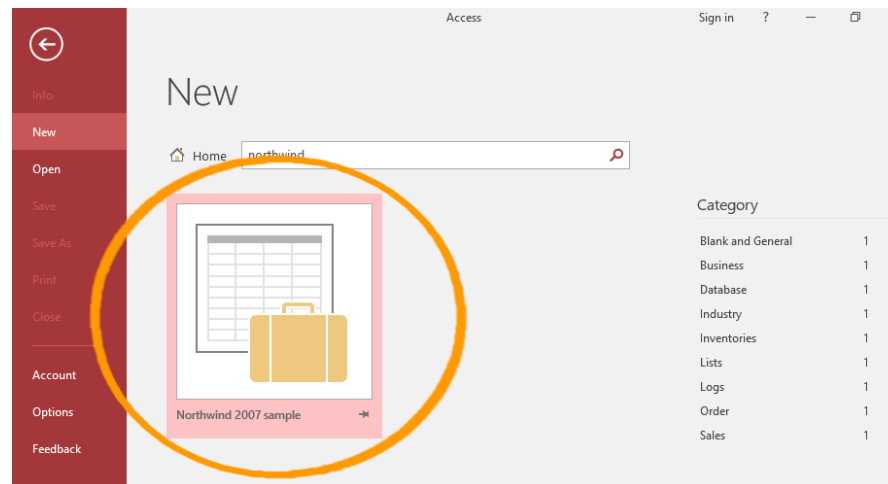
Query1 : Select Query

	Last Name	First Name	Order Date	Quantity
▶	Fuller	Andrew		
	Leverling	Janet	02-May-95	60
	Leverling	Janet	04-Sep-95	40
	Leverling	Janet	07-Apr-95	
	Leverling	Janet	19-Apr-95	30
	Leverling	Janet	19-Apr-95	28
	Leverling	Janet	19-Apr-95	60
	Leverling	Janet	19-Apr-95	30
	Leverling	Janet	24-Apr-95	14
	Leverling	Janet	04-May-95	5
	Leverling	Janet	24-Apr-95	3
	Leverling	Janet	08-Sep-95	5
	Leverling	Janet	02-May-95	20
	Leverling	Janet	09-Aug-95	24
	Leverling	Janet	09-Aug-95	30
	Leverling	Janet	08-Aug-95	25

Record: 1 of 321

# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



Using this database, please try to use different techniques from this practical lesson to create an suitable forms.



# Thank you for your attention!

# ON-LINE DISTANCE COURSE ON DATABASES

## ☐ Module 5. Practice

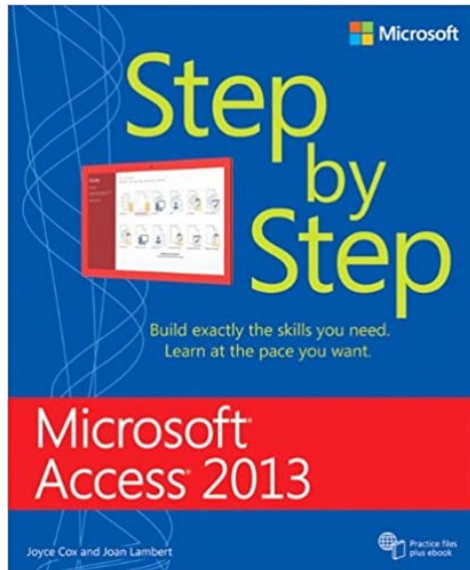
### ☐ Topic 4. Merging of data into one form. Presentation of an effective report.

#### ☐ Practical lesson 2. Presentation of an effective report

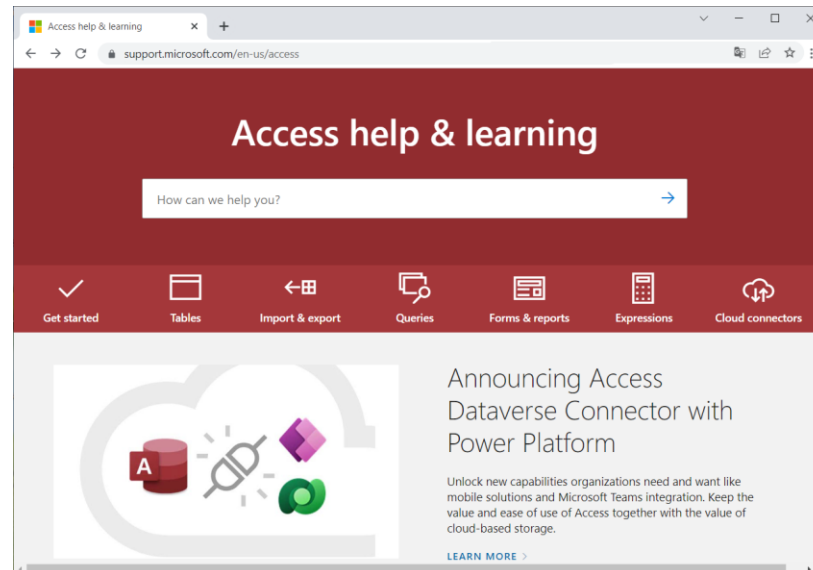




# Practical lesson. Presentation of an effective report.



*Joyce Cox, Joan Lambert.  
Microsoft Access 2013 Step By  
Step, Microsoft Press., 2013.*



*<https://support.microsoft.com/en-us/access>  
<https://www.customguide.com>*

# Introduction to reports in Access

## Creating Reports

If you need to provide information from your database to someone without granting them access to your actual database, you can create a report. Reports enable you to arrange and present your data in a visually appealing format that is easy for the reader to understand. Access provides a user-friendly interface for creating and customizing reports using data from queries or tables in your database.

## Introduction to reports in Access

- Reports offer a way to view, format, and summarize the information in your Microsoft Access database.
- For example, you can create a simple report of phone numbers for all your contacts, or a summary report on the total sales across different regions and time periods.
- From this particular lesson, you'll get an overview of reports in Access. You'll also learn the basics of creating a report, and using options like sorting, grouping, and summarizing the data, and how to preview and print the report.

## Introduction to reports in Access

- A report is a database object that comes in handy when you want to present the information in your database for any of the following uses:
  - Display or distribute a summary of data.
  - Archive snapshots of the data.
  - Provide details about individual records.
  - Create labels.
- The design of a report is divided into sections that you can view in the Design view. Understanding how each section works can help you create better reports. For example, the section in which you choose to place a calculated control determines how Access calculates the results. The following list is a summary of the section types and their uses:

# Introduction to reports in Access

Section	How the section is displayed when printed	Where the section can be used
Report Header	At the beginning of the report.	Use the report header for information that might normally appear on a cover page, such as a logo, a title, or a date. When you place a calculated control that uses the Sum aggregate function in the report header, the sum calculated is for the entire report. The report header is printed before the page header.
Page Header	At the top of every page.	Use a page header to repeat the report title on every page.
Group Header	At the beginning of each new group of records.	Use the group header to print the group name. For example, in a report that is grouped by product, use the group header to print the product name. When you place a calculated control that uses the Sum aggregate function in the group header, the sum is for the current group. You can have multiple group header sections on a report, depending on how many grouping levels you have added. For more information about creating group headers and footers, see the section Add grouping, sorting, or totals.
Detail	Appears once for every row in the record source.	This is where you place the controls that make up the main body of the report.
Group Footer	At the end of each group of records.	Use a group footer to print summary information for a group. You can have multiple group footer sections on a report, depending on how many grouping levels you have added.
Page Footer	At the end of every page.	Use a page footer to print page numbers or per-page information.
Report Footer	At the end of the report. <b>Note:</b> In Design view, the report footer appears below the page footer. However, in all other views (Layout view, for example, or when the report is printed or previewed), the report footer appears <i>above</i> the page footer, just after the last group footer or detail line on the final page.	Use the report footer to print report totals or other summary information for the entire report.

## Create a report in Access

- Step 1: Choose a record source
- The record source of a report can be a table, a named query, or an embedded query. The record source must contain all of the rows and columns of data you want display on the report.
- If the data is from an existing table or query, select the table or query in the Navigation Pane, and then continue to Step 2.
- If the record source does not yet exist, do one of the following:
  - Continue to Step 2 and use the **Blank Report** tool,
  - Or
  - Create the table(s) or query that contains the required data. Select the query or table in the Navigation Pane, and then continue to [Step 2](#).

## Create a report in Access

- Step 2: Choose a report tool
- The report tools are located on the **Create** tab of the ribbon, in the **Reports** group. The following table describes the options:

Tool	Description
Report	Creates a simple, tabular report containing all of the fields in the record source you selected in the Navigation Pane.
Report Design	Opens a blank report in Design view, to which you can add the required fields and controls.
Blank Report	Opens a blank report in Layout view, and displays the Field List from where you can add fields to the report
Report Wizard	Displays a multiple-step wizard that lets you specify fields, grouping/sorting levels, and layout options.
Labels	Displays a wizard that lets you select standard or custom label sizes, as well as which fields you want to display, and how you want them sorted.



## Create a report in Access

- Step 3: Create the report
  1. Click the button for the tool you want to use. If a wizard appears, follow the steps in the wizard and click **Finish** on the last page. Access displays the report in Layout view.
  2. Format the report to achieve the looks that you want:
    - a) Resize fields and labels by selecting them and then dragging the edges until they are the size you want.
    - b) Move a field by selecting it (and its label, if present), and then dragging it to the new location.
    - c) Right-click a field and use the commands on the shortcut menu to merge or split cells, delete or select fields, and perform other formatting tasks.

In addition, you can use the features described in the following sections to make your report more attractive and readable.

## Add grouping, sorting, or totals

- The fastest way to add grouping, sorting, or totals to a desktop database report is to right-click the field to which you want to apply the group, sort, or total, and then click the desired command on the shortcut menu.
- You can also add grouping, sorting, or totals by using the Group, Sort, and Total pane while the report is open in Layout view or Design view:
  1. If the Group, Sort, and Total pane is not already open, on the **Design** tab, in the **Grouping and Totals** group, click **Group & Sort**.
  2. Click **Add a group** or **Add a sort**, and then select the field on which you want to group or sort.
  3. Click **More** on a grouping or sorting line to set more options and to add totals.

# Highlight data with conditional formatting

- Access includes tools for highlighting data on a report. You can add conditional formatting rules for each control or group of controls, and in client reports, you can also add data bars to compare data.
- To add conditional formatting to controls:
  1. Right-click the report in the Navigation Pane and click **Layout View**.
  2. Select the required controls and on the **Format** tab, in the **Control Formatting** group, click **Conditional Formatting**.
  3. In the **Conditional Formatting Rules Manager** dialog box, click **New Rule**.
  4. In the **New Formatting Rule** dialog box, select a value under **Select a rule type**:
    - a) To create a rule that is evaluated for each record individually, select **Check values in the current record or use an expression**.
    - b) To create a rule that compares records to each other by using data bars, click **Compare to other records**.

Under **Edit the rule description**, specify the rule for when the formatting would be applied as well as what formatting should be applied, and then click **OK**.

To create an additional rule for the same control or set of controls, repeat this procedure from step 4.

## Customizing color and fonts

- Try an **App Theme** options to customize the color and fonts.
  1. Open a report in Layout view by right-clicking it in the Navigation Pane and then clicking **Layout View**.
  2. From the **Report Layout Tools** options, on the **Design** tab, click **Themes** and point the cursor over the various themes in the gallery to preview the effects. Click on a theme to select it, and then save your report.
  3. Use the **Colors** or **Fonts** galleries to set colors or fonts independently.

## Add a logo or background image

- You can add a logo or background image to a report and If you update the image, the update is automatically made wherever the image is used in the database.
- To add or remove an image:
  1. In the Navigation Pane, right-click the report and click **Layout View**.
  2. In the report, click the position where you want to add the image and on the **Design** tab, in the **Header/Footer** group, click **Logo**.
  3. Navigate to the image, and click **Open**. Access adds the image to the report.
  4. To remove the image, right-click the image and click Delete from the shortcut menu.
- To add a background image:
  1. In the Navigation Pane, right-click the report and click **Layout View**.
  2. On the **Format** tab, in the **Background** group, click **Background Image**.
  3. Select an image from the **Image Gallery** list or click **Browse**, select an image, and then click **OK**.

## Preview a report

1. Right-click the report in the Navigation Pane and click **Print Preview**. You can use the commands on the **Print Preview** tab to do any of the following:
  - a. Print the report
  - b. Adjust page size or layout
  - c. Zoom in or out, or view multiple pages at a time
  - d. Refresh the data on the report
  - e. Export the report to another file format.
2. Click **Close Print Preview**.

## Print a report

- To print a report without previewing it:
  - Right-click the report in the Navigation Pane and click **Print**. The report is sent to your default printer.

**Note:** *If you select the report in the Navigation Pane and select **Print** from the **File** tab, you can select additional printing options such as number of pages and copies and specify a printer.*

- To open a dialog box where you can select a printer, specify the number of copies, and so on, click **Print**.



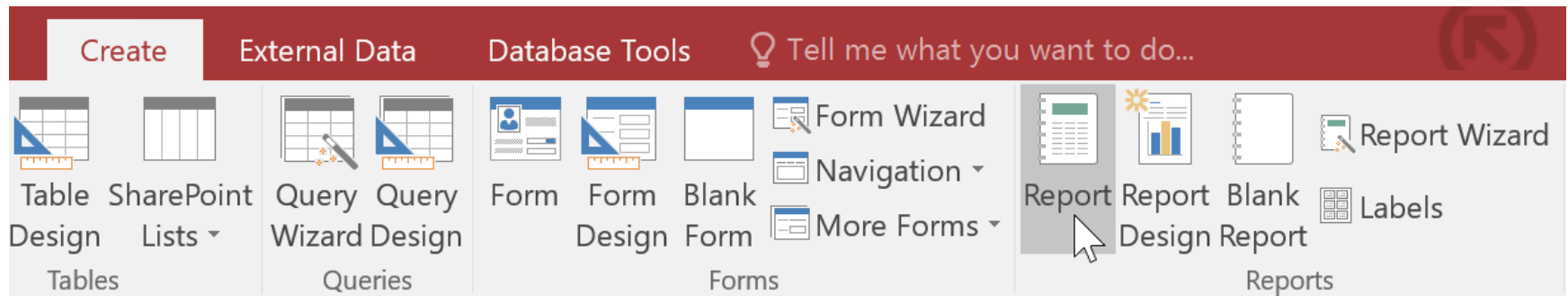
## Example to create a report:

Reports offer the functionality to present sections of your database in a clear and printable layout. Access allows you to generate reports from both tables and queries.

1. Access the table or query that you wish to incorporate into your report. For instance, if we intend to print a list of cookies sold, we will open the Cookies Sold query.

Cookies Sold			
Product Types	Products Table.Product Name	Sales Unit.Product Name	SumOfQuan
Cookies	Fudge Brownie	One Dozen	7
Cookies	Fudge Chocolate	Single	6
Cookies	Ginger Shortbread	One Dozen	6
Cookies	Chocolate Chip	Single	5
Cookies	Butterscotch	Single	3
Cookies	Fudge Brownie	Single	3
Cookies	Cranberry Walnut	One Dozen	3
Cookies	White Chocolate Macademia Nut	Half-Dozen	3
Cookies	Snickerdoodle	Single	3

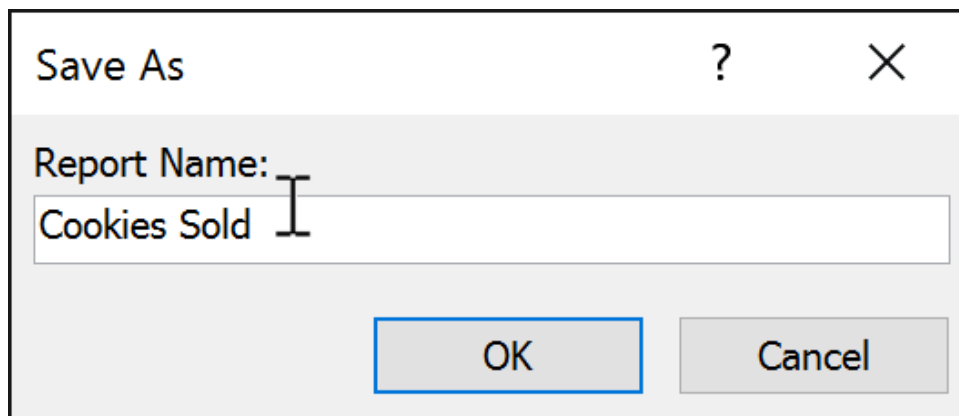
2. Click on the Create tab located in the Ribbon. Find the Reports group and select the Report command.



3. Access will create a new report based on your object.



5. To save your report, simply click on the Save command located on the Quick Access Toolbar. You will be prompted to enter a name for your report, after which you can click OK to complete the saving process.

A screenshot of a 'Save As' dialog box. The title bar says 'Save As' with a question mark and a close button. The main area has a label 'Report Name:' followed by a text input field containing 'Cookies Sold'. At the bottom are 'OK' and 'Cancel' buttons.

Save As ? X

Report Name:

Cookies Sold

OK Cancel



Similar to tables and queries, reports in Access can also be sorted and filtered. You can accomplish this by right-clicking on the field that you want to sort or filter and selecting the appropriate option from the menu that appears.


## Deleting fields

You may discover that your report includes certain fields that are not necessary for viewing. For example, our report includes the Zip Code field, which is not relevant for a list of orders. The good news is that you can remove fields from reports without affecting the table or query from which you extracted the data.

## Example to delete a field in a report:

1. Click on any cell within the field you wish to remove, and then press the Delete key on your keyboard.

 Cookies Sold
  Cookies Sold

 Cookies Sold
 Wednesday,

Product Types	[Products Table].[Product Name]	[Sales Unit].[Product Name]
Cookies	Butter Pecan	One Dozen
Cookies	Butter Pecan	Single
Cookies	Butterscotch	Single
Cookies	Chocolate Banana Walnut	One Dozen
Cookies	Chocolate Banana Walnut	Single
Cookies	Chocolate Chip	Half-Dozen
Cookies	Chocolate Chip	One Dozen
Cookies	Chocolate Chip	Single
Cookies	Cranberry Walnut	One Dozen



When removing a field, make sure to also delete its corresponding header. Select the header and press the Delete key to remove it.

Cookies Sold

---

## Cookies Sold

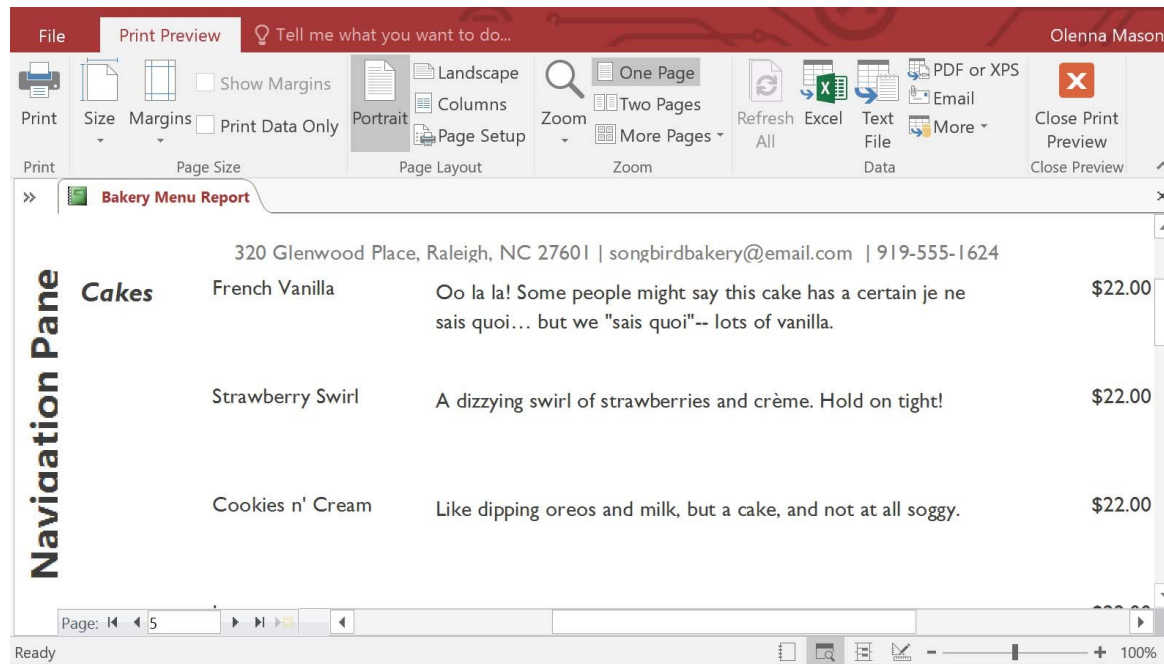
Wednesday,

Product Types	[Products Table].[Product Name]	[Sales Unit].[Product Name]
	Butter Pecan	One Dozen
	Butter Pecan	Single
	Butterscotch	Single
	Chocolate Banana Walnut	One Dozen
	Chocolate Banana Walnut	Single
	Chocolate Chip	Half-Dozen
	Chocolate Chip	One Dozen
	Chocolate Chip	Single
	Cranberry Walnut	One Dozen



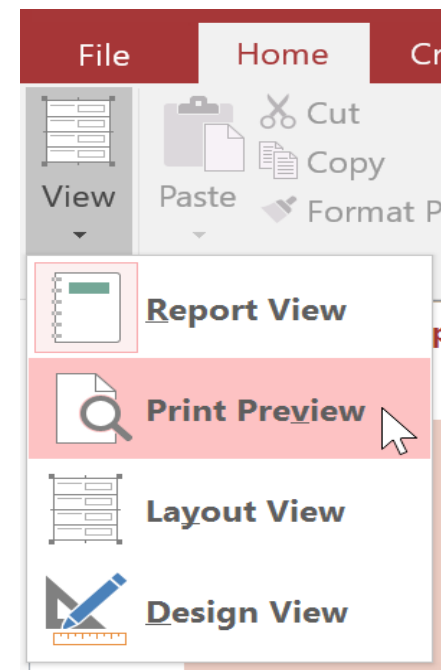
## Example to printing and saving reports in Print Preview:

In addition to using commands in Backstage view to print reports, you can utilize Print Preview. Print Preview provides a preview of your report's appearance on the printed page. It enables you to make adjustments to the display of your report, print it, and even save it in a different file format.



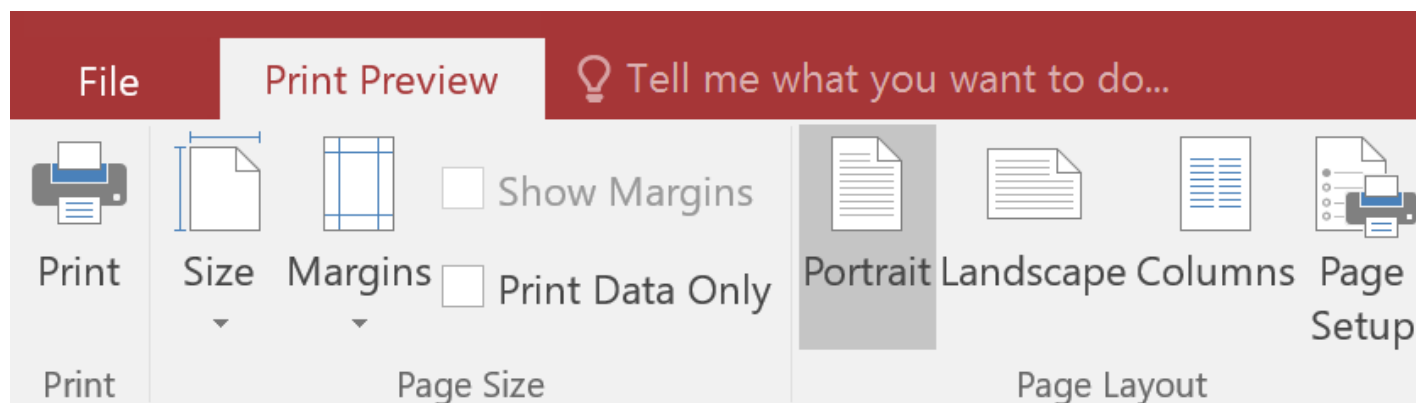
## To print a report:

1. From the Home tab, choose the View option, and then select Print Preview from the available options. This will display your report exactly as it will appear when printed.



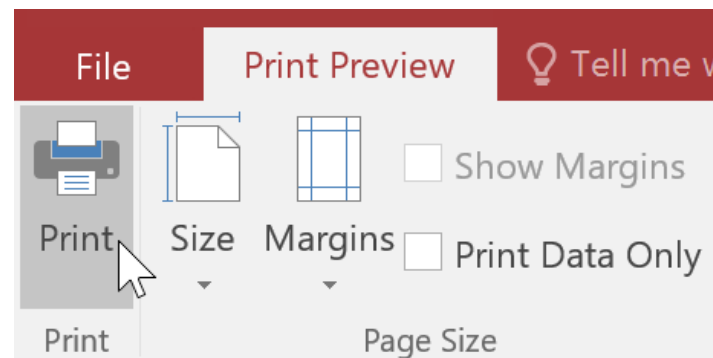
## To print a report:

2. If needed, you can adjust the page size, margin width, and page orientation using the respective commands available on the Ribbon.



## To print a report:

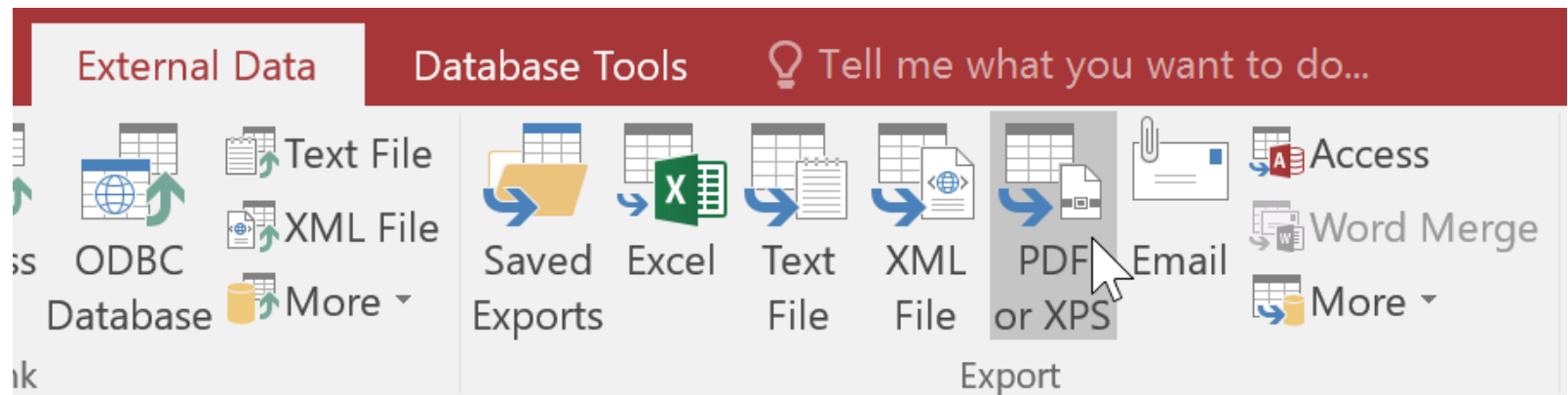
3. Click the **Print** button.



4. The Print dialog box will be displayed. Configure any desired print settings, and then click OK. The report will be printed accordingly.

## To export a report:

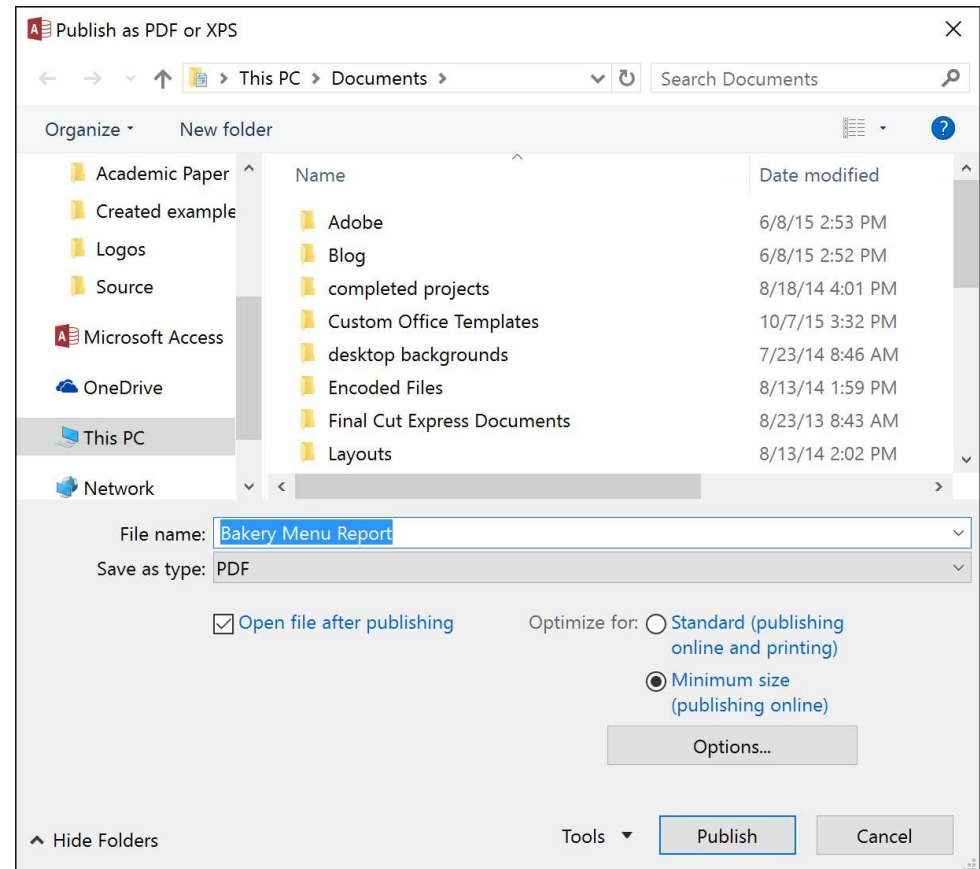
1. Click on the View command in the Home tab and choose Print Preview from the drop-down list.
2. Find the Data group on the Ribbon.
3. Choose one of the available file type options or click More to explore additional options for saving your report as a Word or HTML file.



## To export a report:

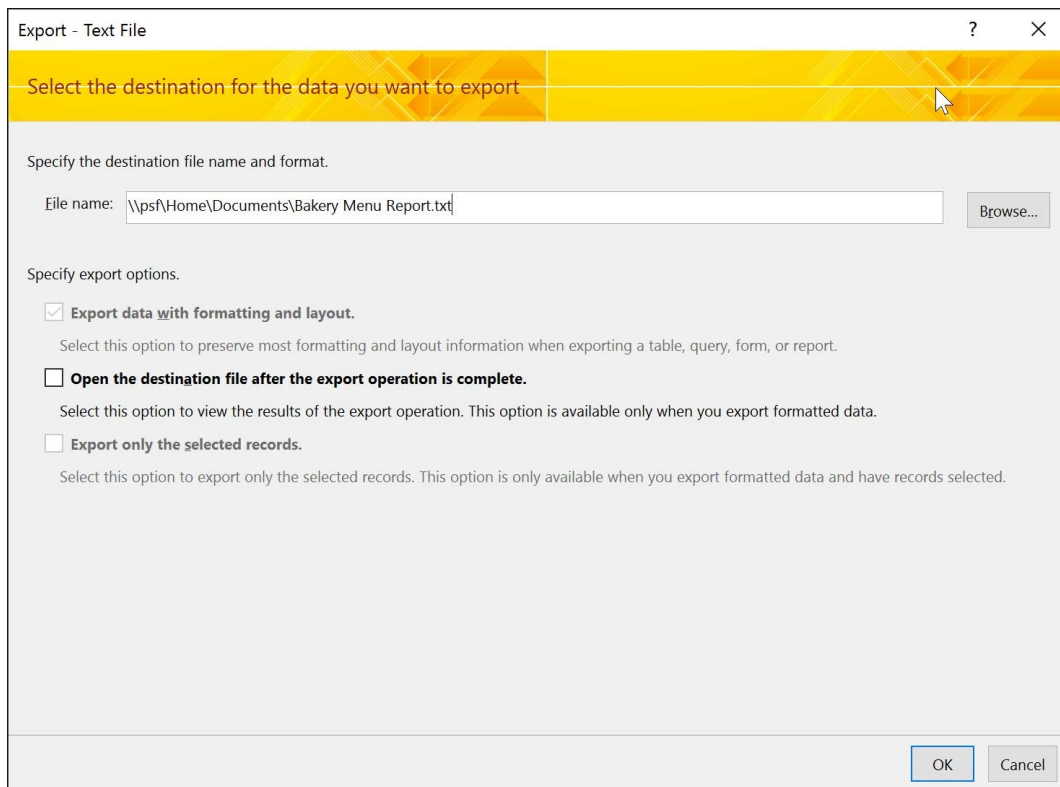
4. A dialog box will be displayed, allowing you to choose the destination where you wish to save the report.

5. Specify a file name for the report and click on the Publish button to proceed.



6. A dialog box will be shown indicating that your file has been saved successfully. Click the Close button to go back to your report.

In some cases, specific export options may trigger the Export Wizard to appear. Simply follow the provided instructions to export your report accordingly.



The dialog box is titled "Export - Text File" and has a yellow header bar with the text "Select the destination for the data you want to export". Below the header, there is a section "Specify the destination file name and format." with a text input field for "File name:" containing the path "\\psf\\Home\\Documents\\Bakery Menu Report.txt" and a "Browse..." button. Below this is a section "Specify export options." with three checkboxes: "Export data with formatting and layout." (checked), "Open the destination file after the export operation is complete." (unchecked), and "Export only the selected records." (unchecked). Each checkbox has a descriptive text below it. At the bottom right, there are "OK" and "Cancel" buttons.

Export - Text File

Select the destination for the data you want to export

Specify the destination file name and format.

File name: \\psf\\Home\\Documents\\Bakery Menu Report.txt Browse...

Specify export options.

☒ Export data with formatting and layout.  
Select this option to preserve most formatting and layout information when exporting a table, query, form, or report.

☐ Open the destination file after the export operation is complete.  
Select this option to view the results of the export operation. This option is available only when you export formatted data.

☐ Export only the selected records.  
Select this option to export only the selected records. This option is only available when you export formatted data and have records selected.

OK Cancel



## Guide to designing effective reports

- When you design a report, you must first consider how you want the data arranged on the page and how the data is stored in the database.
- During the design process, you might even discover that the arrangement of data in the tables will not allow you to create the report that you want.
- This can be an indication that the tables are not normalized — this means that the data is not stored in the most efficient manner.

# Guide to designing effective reports

- Make a sketch of your report
  - This step is not required — you might find that the Access Report Wizard or the Report tool (both of which are available on the Create tab, in the Reports group) provide a sufficient starting design for your report. However, if you decide to design your report without using these tools, you might find it helpful to make a rough sketch of your report on a piece of paper by drawing a box where each field goes and writing the field name in each box. Alternatively, you can use programs such as Word or Visio to create a mockup of the report. Whichever method that you use, be sure to include enough rows to indicate how the data repeats.

# Guide to designing effective reports

- Make a sketch of your report



For example, you can use a row for product information, then several repeating rows for that product's sales, and finally a row of sales totals for the product. Then, the sequence repeats for the next product and so on until the end of the report. Or, perhaps your report is a simple listing of the data in the table, in which case your sketch can contain just a series of rows and columns.

# Guide to designing effective reports

- Make a sketch of your report
  - After you create your sketch, determine which table or tables contain the data that you want to display on the report. If all the data is contained in a single table, you can base your report directly on that table. More often, the data that you want is stored in several tables that you must pull together in a query, before you can display it on the report. The query can be embedded in the **RecordSource** property of the report, or you can create a separate, saved query and base the report on that.

# Guide to designing effective reports

- Decide which data to put in each report section
  - Each report has one or more report sections. The one section that is present in every report is the Detail section. This section repeats once for each record in the table or query that the report is based on. Other sections are optional and repeat less often and are usually used to display information that is common to a group of records, a page of the report, or the entire report.
- Decide how to arrange the detail data
  - Most reports are arranged in either a tabular or a stacked layout, but Access gives you the flexibility to use just about any arrangement of records and fields that you want.
  - Tabular layout A tabular layout is similar to a spreadsheet. Labels are across the top, and the data is aligned in columns below the labels.

# Guide to designing effective reports

- Decide how to arrange the detail data



**Employees**

ID	Last Name	First Name	Job Title	Business Phone
1	Freehafer	Nancy	Sales Representative	(123)456-7890
2	Cencini	Andrew	Vice President, Sales	(123)456-7890
3	Kotas	Jan	Sales Representative	(123)456-7890
4	Sergienko	Mariya	Sales Representative	(123)456-7890
5	Thorpe	Steven	Sales Manager	(123)456-7890
6	Neipper	Michael	Sales Representative	(123)456-7890

Tabular refers to the table-like appearance of the data. This is the type of report that Access creates when you click **Report** in the **Reports** group of the **Create** tab. The tabular layout is a good one to use if your report has a relatively small number of fields that you want to display in a simple list format. The following illustration shows an employee report that was created by using a tabular layout.

# Guide to designing effective reports

- Decide how to arrange the detail data

**Stacked layout** A stacked layout resembles a form that you fill out when you open a bank account or make a purchase from an online retailer. Each piece of data is labeled, and the fields are stacked on top of each other. This layout is good for reports that contain too many fields to display in a tabular format — that is, the width of the columns would exceed the width of the report. The following illustration shows an employee report that was created by using a stacked layout.



**Employees**

ID	1
Last Name	Freehafer
First Name	Nancy
Job Title	Sales Representative
Business Phone	(123)456-7890

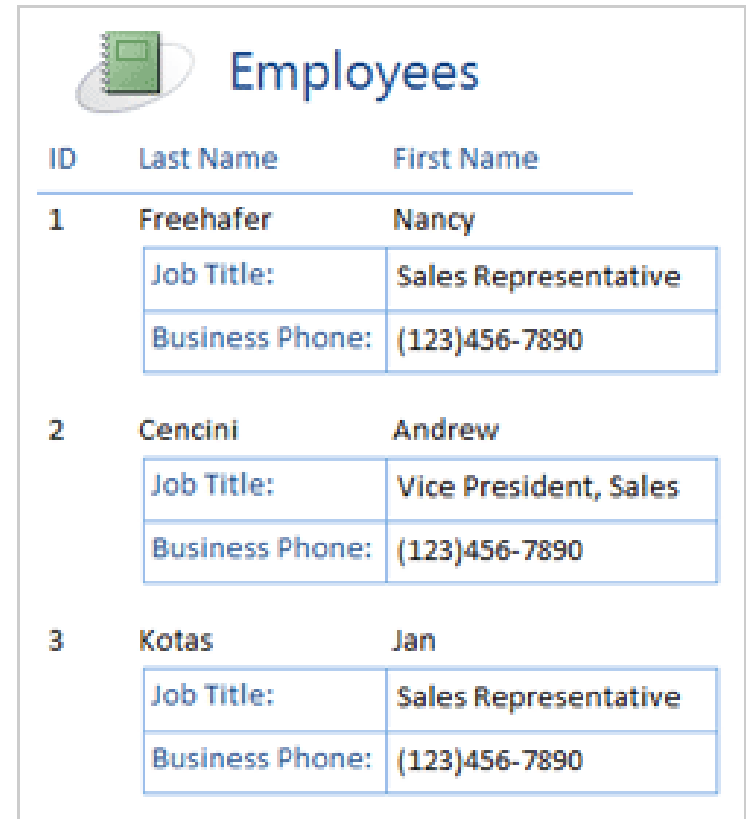
ID	2
Last Name	Cencini
First Name	Andrew
Job Title	Vice President, Sales
Business Phone	(123)456-7890



# Guide to designing effective reports

- Decide how to arrange the detail data

**Mixed layout** You can mix elements of tabular and stacked layouts. For example, for each record, you can arrange some of the fields in a horizontal row at the top of the Detail section and arrange other fields from the same record in one or more stacked layouts beneath the top row. The following illustration shows an employee report that was created by using a mixed layout. The ID, Last Name, and First Name fields are arranged in a tabular control layout, and the Job Title and Business Phone fields are arranged in a stacked layout. In this example, gridlines are used to provide a visual separation of fields for each employee.



The illustration shows a report titled "Employees" with a green folder icon. It displays three employee records. Each record has a tabular header row with fields: ID, Last Name, and First Name. Below the header, the Job Title and Business Phone fields are arranged in a stacked layout. Gridlines are used to separate the fields for each employee.

ID	Last Name	First Name
1	Freehafer	Nancy
	Job Title:	Sales Representative
	Business Phone:	(123)456-7890
2	Cencini	Andrew
	Job Title:	Vice President, Sales
	Business Phone:	(123)456-7890
3	Kotas	Jan
	Job Title:	Sales Representative
	Business Phone:	(123)456-7890

# Guide to designing effective reports

- Decide how to arrange the detail data

## Employees

ID	Company	Last Name	First Name
1	Northwind Traders	Freehafer	Nancy
E-mail Address	Job Title	Business Phone	Home Phone
nancy@northwindtraders.com	Sales Representative	(123)456-7890	(123)456-7890
Mobile Phone	Fax Number		
	(123)456-7890		
Address			
123 Any Street			
City	State/Province	ZIP/Postal Code	Country/Region
Any City	WA	99999	USA

ID	Company	Last Name	First Name
2	Northwind Traders	Cencini	Andrew
E-mail Address	Job Title	Business Phone	Home Phone
andrew@northwindtraders.com	Vice President, Sales	(123)456-7890	(123)456-7890
Mobile Phone	Fax Number		
	(123)456-7890		
Address			
123 Any Street			

**Justified layout** If you use the Report Wizard to create your report, you can choose to use a justified layout. This layout uses the full width of the page to display the records as compactly as possible. Of course, you can achieve the same results without using the Report Wizard, but it can be a painstaking process to align the fields exactly. The following illustration shows an employee report that was created by using the Report Wizard's justified layout.

# Use control layouts to align your data

Control layouts are guides that you can add to a report while it is open in Layout view or Design view. Access adds control layouts automatically when you use the Report Wizard to build a report, or when you create a report by clicking **Report** in the **Reports** group of the **Create** tab. A control layout is like a table, each cell of which can contain a label, a text box, or any other type of control. The following illustration shows a tabular control layout on a report.

Last Name	First Name	Business Phone
Cencini	Andrew	(123)456-7890
Freehafer	Nancy	(123)456-7890
Giussani	Laura	(123)456-7890
Hellung-Larsen	Anne	(123)456-7890
Kotas	Jan	(123)456-7890
Neipper	Michael	(123)456-7890
Sergienko	Mariya	(123)456-7890
Thorpe	Steven	(123)456-7890
Zare	Robert	(123)456-7890

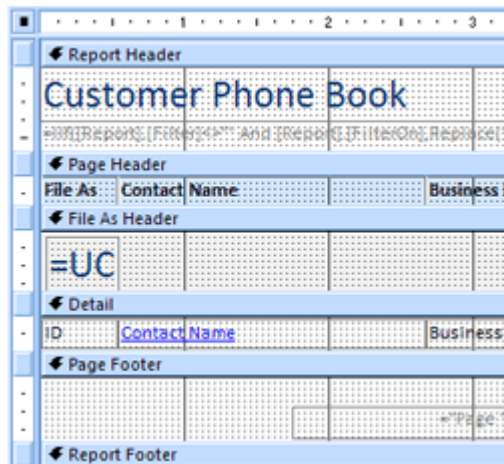
The orange lines indicate the rows and columns of the control layout, and they are visible only when the report is open in Layout view or Design view. Control layouts help you achieve a uniform alignment of data in rows and columns, and they make it easier to add, resize, or remove fields. By using the tools in the **Table** and **Position** groups on the **Arrange** tab (available in Layout view or Design view), you can change one type of control layout to another, and you can remove controls from layouts so that you can position the controls wherever you want on the report.

## Add or remove report or page header and footer sections

- The headers and footers are report sections that you can use to display information that is common to the entire report, or to each page of a report.
- For example, you can add a Page Footer section to display a page number at the bottom of each page, or you can add a Report Header section to display a title for the entire report.
- **Add report or page header and footer sections:**
  1. In the Navigation Pane, right-click the report that you want to change, and then click **Design View** on the shortcut menu.
  2. Verify which sections are already on the report. The sections are separated by shaded horizontal bars called **section selectors**. The label on each section selector indicates what the section directly below it is.

# Add or remove report or page header and footer sections

- Add report or page header and footer sections:



Report Header			
Customer Phone Book			
Page Header			
File As	Contact Name		Business
File As Header			
=UC			
Detail			
ID	Contact Name		Business
Page Footer			
			Page
Report Footer			

Every report has a Detail section and can also contain Report Header, Page Header, Page Footer, and Report Footer sections. In addition, if there are grouping levels in the report, you might see group headers or footers (such as the **File As Header** shown in the preceding illustration). By default, group headers and footers are named by using the field name or expression that is the basis of the group. In this case, the name of the grouping field is "File As."

- To add page header and footer sections or report header and footer sections to your report, right-click any section selector and then click **Page Header/Footer** or **Report Header/Footer** on the shortcut menu.

# Add or remove report or page header and footer sections

- Remove report or page header and footer sections
1. In the Navigation Pane, right-click the report that you want to change, and then click **Design View** on the shortcut menu.
  2. Right-click any section selector and then click **Page Header/Footer** or **Report Header/Footer** on the shortcut menu.

If you are removing a header and footer pair and those sections contain controls, Access warns you that deleting the sections will also delete the controls and that you will not be able to undo the action.

Click **Yes** to remove the sections and delete the controls, or click **No** to cancel the operation.



# Tips for formatting different data types

- When you create a report by using the **Report** tool (available on the **Create** tab, in the **Reports** group), or by using the Report Wizard, Access adds the fields to the report for you and creates the most appropriate control to display each field, based on the field's data type.
- If you are adding fields to a report yourself, the preferred method is to drag each field from the **Field List** to the report.
- As with the Report Wizard or the **Report** tool, Access creates the most appropriate control for each field, depending on the field's data type. For most data types, the most appropriate (default) control to use is the text box.
- The following sections provide tips about how to format some of the special case data types.
- **Multivalued fields** The default control for a multivalued field is a combo box. This can seem like a strange choice for a control on a report, because you can't click the arrow on a combo box in a report. However, in the context of a report, a combo box behaves like a text box. The arrow is visible only in Design view.



# Tips for formatting different data types

- If the field contains multiple values, those values are separated by commas. If the combo box is not wide enough to display all the values on one line and the **CanGrow** property of the combo box is set to **Yes**, the values wrap to the next line.
- Otherwise, the values are truncated. To set the **CanGrow** property for a control, open the report in Design view or Layout view, click the control, and then press F4 to display the control's property sheet. The **CanGrow** property is located on both the **Format** tab and the **All** tab of the property sheet for the control.
- **Rich text fields** The default control for a rich text field is a text box. If the text box is not wide enough to display all the values on one line and the **CanGrow** property of the text box is set to **Yes**, the values wrap to the next line.
- Otherwise, the values are truncated. To set the **CanGrow** property for a control, open the report in Design view or Layout view, click the control, and then press F4 to display the control's property sheet. The **CanGrow** property is located on both the **Format** tab and the **All** tab of the property sheet for the control.

# Good sample report

## Customer Orders

### Big Toys LLC

Order Date	Product	Payment Status	Quantity	Unit Price	Discount	Line Total
04-30-10	Converter Boxes	Invoice Sent	5	\$35.00	5.00%	\$166.25
11-24-10	Internet Products		1	\$74.99	4.29%	\$71.77
11-24-10	Converter Boxes		0	\$99.00	0.00%	\$0.00
11-24-10	Converter Boxes		5	\$49.99	4.29%	\$239.23
11-24-10	Accessories		2	\$35.00	0.00%	\$70.00
04-30-10	DVR Player	Invoice Sent	2	\$150.00	2.00%	\$294.00
03-01-10	DVR Player	Invoice Sent	1	\$150.00	5.00%	\$142.50
03-01-10	Internet Products	Invoice Sent	2	\$59.99	0.00%	\$119.98
						<b>\$1,103.73</b>

Total Big Toys LLC records: 8

### Corrinne Davies Family Practice

Order Date	Product	Payment Status	Quantity	Unit Price	Discount	Line Total
01-01-06	DVR Player	Paid in Full	1	\$150.00	10.00%	\$135.00
02-22-08		Paid in Full				
01-01-06	Converter Boxes	Paid in Full	5	\$35.00	20.00%	\$140.00
						<b>\$275.00</b>

Total Corrinne Davies Family Practice records: 3

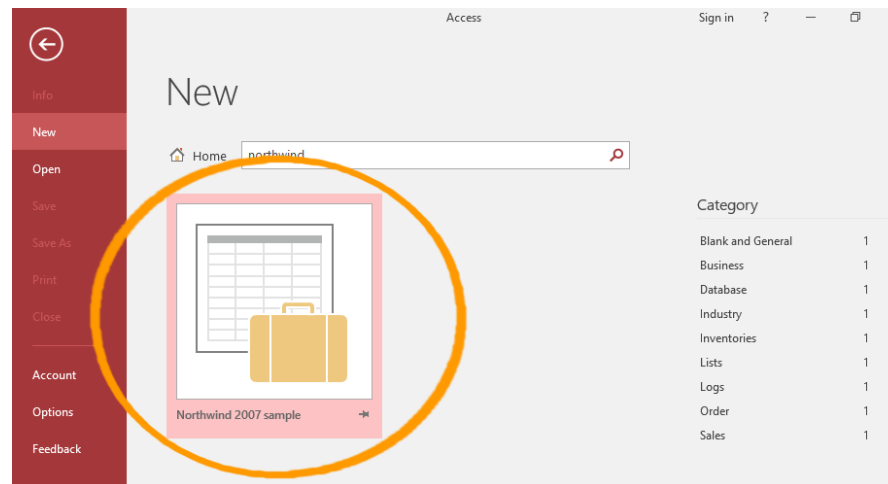
George P. James

ERASMUS+

Key Action KA2 - Cooperation for innovation and the exchange of good practices  
Action Type KA226 - Partnerships for Digital Education Readiness

# Exercises

- The Northwind database is a sample database, designed to assist in learning and demonstrations, etc. It demonstrates what an inventory/orders system might look like for a mail order dry goods company.
- The Northwind sample database is based on a fictitious company called Northwind Traders, which imports and exports specialty foods from around the world.



Using this database, please try to use different techniques from this practical lesson to create an effective report.



# Thank you for your attention!