DIUM, University of Udine

Basi di dati
Introduction

Andrea Brunello
andrea.brunello@uniud.it

- Preliminaries
  - *Some basic mathematical notions*
- The need for databases
  - *Why can't we simply store files on the disk?*
- Abstraction levels and data models
  - *Things that allow us to define and organize data*
- Relational databases
  - *The evergreen way-to-go for databases*
- Database internals
  - *How is data stored? How can we interact with a database?*
- Database design
  - *The steps to follow to build a database*
- History of information systems
 - *When did it all start?*

- In mathematics, a collection of elements represents a set if there is an objective criterion that makes it possible to unambiguously decide whether any element is part of the grouping or not

- For example: the set of natural numbers, the set of world capital names

- The objects that make up a set are called elements of that set

- Notation: *A* set, *a* element of set *A*. We denote set membership with $a \in A$

 - Example: $A = \{7, 1, 3, 2, 5\}$, then $1 \in A$

A set is characterised by the following properties:

- an element may or may not belong to a given set, there is no third option

- an element may not appear more than once in a set

- the elements of a set have no order in which they appear

- the elements of a set characterise it uniquely: two sets coincide if and only if they have the same elements
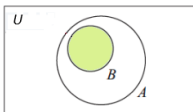
Set Operations and Venn Diagrams
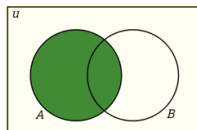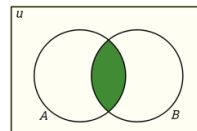

Set A
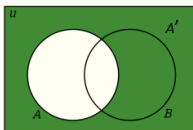
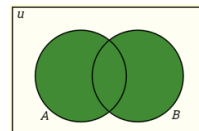A' the complement of A

A and B are disjoint sets

B is proper subset of A $B \subset A$

Both A and B $A \cap B$
A intersect B

Either A or B $A \cup B$
A union B

- **Multiset**: generalisation of the concept of a set in which repeated elements are allowed

  - For example: $A = \{1, 1, 3, 3, 2, 3\}$

- **Cartesian product**: given two sets A e B, the cartesian product of A and B, denoted with $A \times B$, is given by the set of ordered pairs (a, b) where $a \in A$ and $b \in B$

  - For example: $A = \{1, 2, 3\}$, $B = \{red, blue\}$, then
    $A \times B = \{(1, red), (1, blue), (2, red), (2, blue), (3, red), (3, blue)\}$

- Regardless of the domain considered, data are the lifeblood of corporate information systems

- **Data** can be defined as a collection of raw, unorganised and unanalysed material relating to a phenomenon (e.g., a thermometer outputting decimal numbers)

- Through a processing and interpretation phase, the data become meaningful **information** for the recipient (e.g., the numbers are interpreted as a temperature in degrees $°C$)

- Combining information, intuition and experience yields **knowledge**, which can be compared with knowledge already acquired and which allows us to interpret and guide our actions

Database Management System (DBMS)

- A DBMS contains information on a particular domain
  - Collection of interrelated data (database)
  - Set of programs to manage the data:
    - Definition of structures to store data
    - Mechanisms to retrieve/modify data
    - Safety and concurrency aspects

- Database applications:
  - Banking: bank transfers, interactions with the ATMs
  - Airlines: reservations, schedules
  - Universities: student registrations, grade assignment
  - Sales: customers, inventory, products and sales
  - E-commerce: e.g, when you buy something from Amazon

- In general, databases can be very large

- They touch all aspects of our lives, although user interfaces
  hide access details

- A University database could support interactions such as:
    - Add new students, instructors, and courses
    - Register students for courses, and generate class rosters
    - Plan exams, assign grades to students, compute grade point averages

- In the early days, such applications were built directly on top of the file system

- Ad-hoc solutions: data stored in files (e.g., .csv) and applications coded in scripts
    - Over time, they both tend to grow in their number

- First relational database in early 80s, after 10 years of work
    - So, the need for a DBMS was seen from the beginning

Actors in Things I've Seen Recently.csv - Notepad

File   Edit   Format   View   Help

```
Name,Birthdate,Birthplace,Sex
Chris Pratt,6/21/79,"Virginia, MN, USA",M
Ellen Barkin,4/16/54,"New York City, New York, USA",F
Lee Byung-hun,8/13/70,"Seongnam, South Korea",M
Eduardo Noriega,8/1/73,"Santander, Cantabria, Spain",M
Michael Dorman,4/26/81,"Auckland, New Zealand",M
Emily Blunt,2/23/83,"London, England, UK",F
Frances McDormand,6/23/57,"Gibson City, IL, USA",F
Ron Livingston,6/5/67,"Cedar Rapids, IA, USA",M
Morgan Freeman,6/1/37,"Memphis, TX, USA",M
Noomi Rapace,12/28/79,"Hudiksvall, Sweden",F
Djimon Hounsou,4/24/64,"Cotonou, Benin",M
Natalia Reyes,2/6/87,"Bogota, Columbia",F
```

Ln 1, Col 1   100%   Windows (CRLF)   UTF-8

Drawbacks of using file system to store data

- Data redundancy
  - Multiple file formats and programming languages
  - Duplication of information in different files, e.g., used by different departments (teaching office, payroll office, . . .)
  - Other than wasting memory, brings to inconsistencies

- Difficulty in accessing data
  - Need to write a new program to carry out each new task
    - E.g., a program to get info about all professors of Computer Science, another for those teaching Economy, . . .
    - Also, data access is not "interactive"

- Data isolation
  - Multiple files and formats are difficult to manage

- Constraints management
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather being stated explicitly
  - Hard to add new constraints or change existing ones

- Atomicity of updates
  - Failures may leave the database in an inconsistent state with partial updates carried out
  - E.g., transfer of funds from one account to another should either complete or not happen at all

- Concurrent access by multiple users
  - What happens if two users try to access and modify the same data?
  - E.g., two users booking the last airline ticket
  - For efficiency reasons, we must support concurrency

- Security problems
  - Hard to provide a user access to some, but not all, data

⤳ DBMSs provide solutions to all these problems

- Large amounts of data
  - Not possible to load all of them into *main memory*
  - Critical complexity factor for a DBMS is the number of transfers to/from disk

- Global data
  - DB and its data are relevant for a vast array of users and applications
  - They are typically not tied to a particular user or application, as standard programs are

- Persistence of data
  - DB is there independently from the interacting users or applications
 - Data lives on its own unlike, for instance, program variables

Abstractions

- To tackle the complexity inherent in their development and usage, DBMSs make use of **abstractions**

- Abstractions are a fundamental concept in computer science, for instance, in the Operating Systems domain

- They keep complexity under control allowing the user to focus on what is important, leaving out irrelevant details that would generate confusion
  - It is like flying on a plane at different altitudes, or looking at a painting at different distances
  - When you are driving your car, you do not care about how the engine is working (hopefully)

- In practice, for what concerns the database realm, it involves stratifying the DBMS
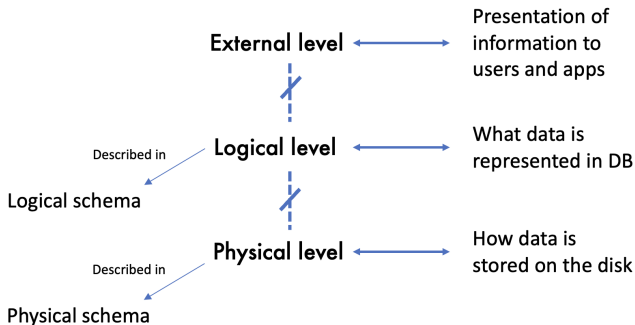
DBMS levels of abstraction

- From lowest to highest:
  - **Physical level:** *how* data is stored on the disk (data structures, file arrangement, . . .)
  - **Logical level:** *what* data is stored and represented in the database and their relationships
    - describes an entire database in terms of a small number of relatively simple structures
    - e.g., by means of tables
    - they are *tables*, we do not care about how they are stored on the disk (the physical level deals with it)
  - **View/External level:** manages the presentation of information to users and programs. Views can also hide information to some kinds of users (e.g., show only some parts of a table)

- Each level encapsulates/hides the details it deals with, and can take advantage of the functionalities made available by lower levels, without caring about their implementation
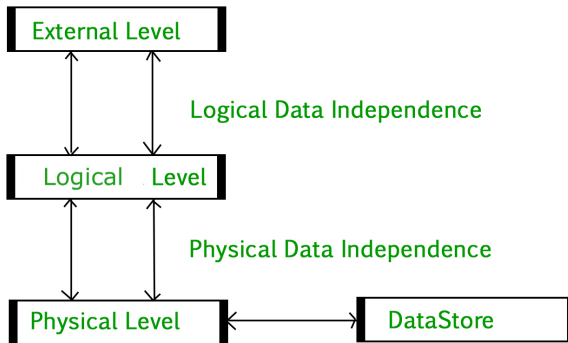
- **Physical data independence:** ability to make changes to the physical level without having to modify the logical one

- It's a concept similar to the APIs (Application Programming Interface)

- The levels of abstraction of a database are described by means of *schemas*:
    - **Physical schema:** it lays out how data are stored physically on a storage system in terms of files and indices
    - **Logical schema:** it encodes the overall logical structure of the database
        - E.g., the database consists of information about a set of customers (each characterized by name, address, and phone number) and their accounts in a bank; the relationships among them; and, some constraints over them
        - It describes, in a form understandable to the system, the characteristics of the elements I am interested in dealing with

- While the schemas lay down the "structure of the database" at different levels, the **instance** of the database refers to its actual content at a particular moment
    - E.g., the specific, concrete customers and accounts

External level → Presentation of information to users and apps

Logical level → What data is represented in DB

Described in → Logical schema

Physical level → How data is stored on the disk
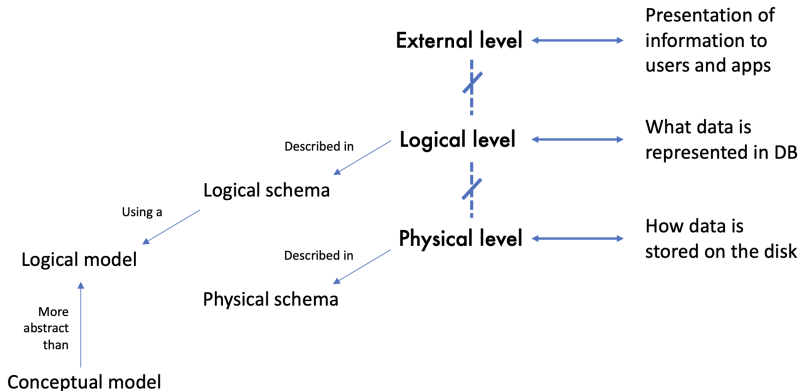
Described in → Physical schema

Data models

- To the describe the logical schema, a small number of relatively simple structures is used (data model)

- A **data model** is a collection of tools for describing:
  - Data (and its structure)
  - Data relationships
  - Data constraints

- There exist several data models
  - The **relational data model**, which considers records and relations (rows and tables)
  - Graph data model, that deals with nodes and vertices
  - Object-based data models
  - Semistructured data models (XML)
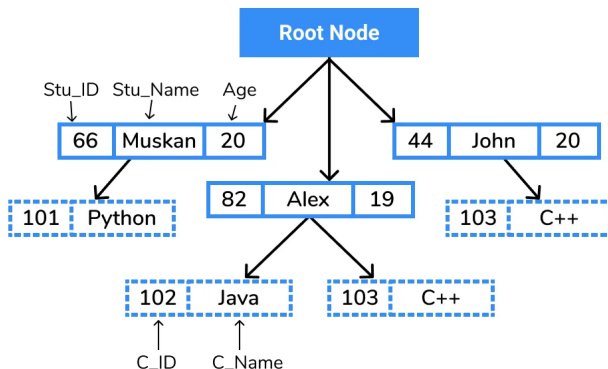 - Older models: Hierarchical model, Network model

- *Logical data models* are abstract: they describe data irrespective from how they will be physically stored

- Still, they reflect a specific organization of the data (using tables, graphs, etc)

- **Conceptual data models** are even more abstract, and are used to describe the data irrespective from the choice of the logical model
  - They are extremely useful during the first phases of the database design process
  - In this course, we will talk about the Entity-Relationship model

External level ←————→ Presentation of information to users and apps

Logical level ————→ What data is represented in DB

Described in ← Logical schema

Using a ← Logical model

Physical level ←————→ How data is stored on the disk

Described in ← Physical schema

More abstract than

Conceptual model

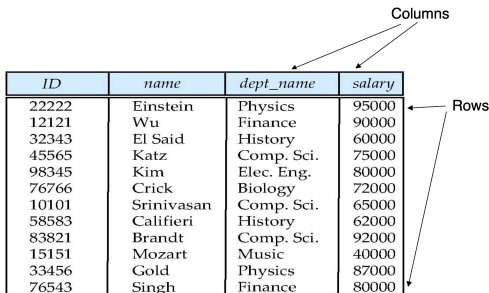Records physically point to other records; only "parent-child" relationships are allowed
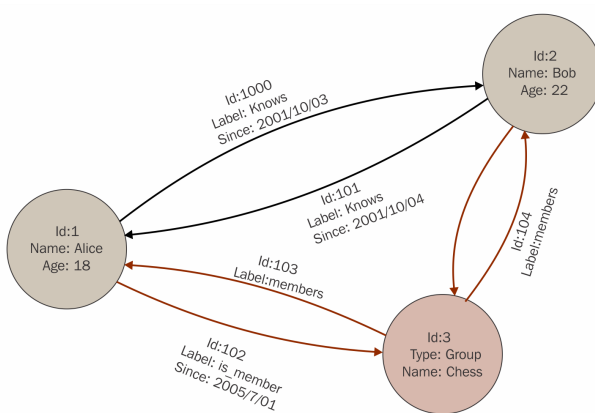
Records physically point to other records; more flexible structure for the nodes and the relationships

- In the relational model, all data is stored by means of tables
- Such tables represent both data and the relationships among data
- It is a kind of **record-based** model: rows have a fixed format and length
- Each table has a **primary key**

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Columns

Rows

(a) The *instructor* table

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	building	budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- There are row insert/update/delete anomalies, for instance:
    - Cannot insert a professor without a linked department
    - Cannot keep a department if it does not have any professors
    - Potential need to modify multiple rows to update the budget of a department

- Intuitively, the problem is that the table does not have a clear semantics
    - It is mixing information about professors and departments
    - The typical solution involves splitting the table

- **Normalization theory** provides formal techniques to prevent this kind of problems

Structured Query Language

- A very nice thing about the relational DBs is that they come with SQL (Structured Query Language)

- SQL is a *declarative* language composed of two parts:
  - DDL (Data Definition Language): that allows to specify the schema of the database
  - DML (Data Manipulation Language): that allows to interact with the content of the database (i.e., the database instance) by means of *queries*

- SQL features a very intuitive syntax

- It is standardized within ISO/IEC, so to allow for a better interoperability between different relational DBMSs

SELECT name
FROM instructor
WHERE salary > 60000;
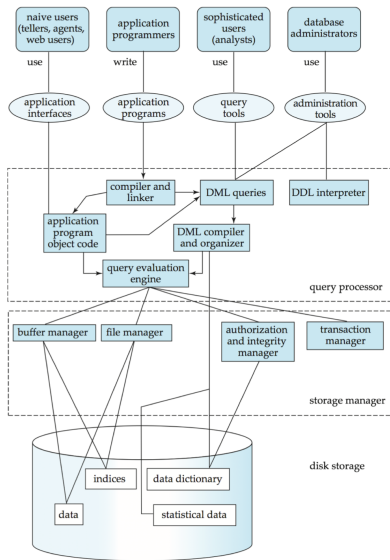
- Other than data itself, the database contains several other information:

  - **Indices:** auxiliary structures that speed up data access when certain conditions are satisfied

  - **Data dictionary:** metadata that define the structure of the database

  - **Statistical data:** regarding the content of the database, the distribution of data, etc.

Database engine

- From the point of view of its implementation, the DBMS is composed of a set of sub-modules, each dealing with a specific aspect of data and their access

  - Storage manager

  - Query processor

  - Transaction manager

Storage manager

- The storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system

- It is in charge of the following tasks:
  - Interaction with the OS file manager
  - Efficient storage, retrieval, and update of data
  - In the case of SQL, translation of DML statements into low-level file-system commands

- Deals with:
  - Storage access
  - File organization
  - Indexing

- Steps:
  1. Parsing and translation
  2. Optimization
  3. Evaluation
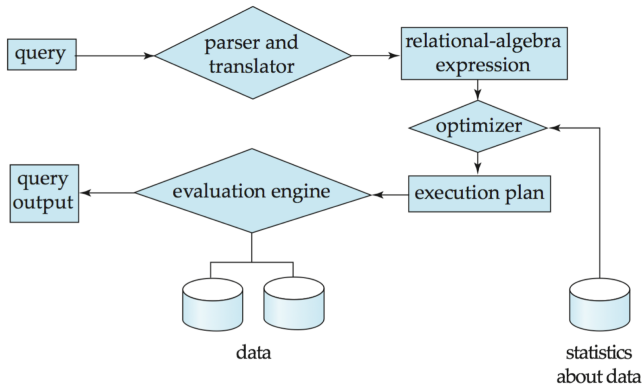
- In general, there are many alternative ways of evaluating a given query
  - Equivalent expressions: e.g., first select the row with *ID* = *22222* from *instructor*, then remove all columns except for *ID* and *name*, or vice-versa
  - Different algorithms for each operation: e.g., perform a table scan or rely on indexes

- The cost difference (in terms of execution time and disk transfers) between a good and a bad way of evaluating a query can be enormous

- Need to estimate the cost of operations:
  - Heavily depends on statistical information about relations which the database must maintain (e.g., number of rows in a table)
  - Need to estimate statistics for intermediate results to compute the cost of complex expressions

- What if the system fails?

- What if more than one user is concurrently updating the same data?

- A **transaction** is a collection of operations that performs a single logical function in a database application
  - E.g., a money transfer between two bank accounts

- The transaction manager ensures that the DB remains in a consistent state despite of system failures (e.g., power failures or OS crashes) and transaction failures

- In addition, interactions among concurrent transactions is monitored, to ensure the consistency of the database
  - We do not want transactions to interfere with one another in undesired ways

- The architecture of a database system is greatly influenced by the underlying computer system on which the database is running:
  - Centralized
  - Client-server
  - Parallel (multi-processor)
  - Distributed

- In this course we will not deal with these aspects

The process of designing a database is typically articulated into different phases, that separate *what* to represent from *how* to represent it:

1. **Conceptual design**

2. **Logical design**

3. **Physical design**

At each stage, we lose expressiveness but gain formality.

- The main aim is to identify and describe the domain of interest, that should be modeled by the database, along with all the requirements

- Typically carried out through a series of brainstorming meetings with IT personnel and all interested stakeholders

- As a result, a *conceptual schema* that makes use of a *conceptual model* is developed (e.g., practically an Entity-Relationship diagram may be drawn)

- The designer must try to represent the information content of the database, without worrying about how this information will be encoded in a real system and which kinds of operations will be carried out over the data

- Translation of the conceptual schema into a *logical schema*, that refers to a specific *logical model* (e.g., the relational model)

- Typically, a set of restructuring and translation rules is followed, that also take into into account which kind of operations are going to be performed on the data

- Also, formal techniques to ensure the quality of the logical schema are employed, such as *normalization techniques* for the relational model

- At this stage, a specific DBMS technology must be chosen (e.g., relational or graph DB)

- Enrichment of the logical schema with details regarding the physical, low-level aspects

- For instance, indexes to speed up the access to the data may be defined

- The *physical schema* is thus obtained

History of information systems

- Every kind of organization has an *information system* which organizes and manages data useful to support relevant business goals

- Information systems pre-date computers and even electricity, and are not necessarily automated systems

- For instance, think about accounting and bookkeeping, or the civil registry
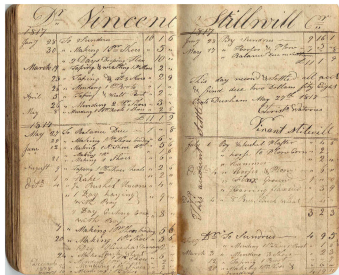
- 1950s and early 1960s:
  - Data processing done using magnetic tapes for storage, which provided only *sequential access*
  - Punched cards for input
  - Support for tasks such as payroll management
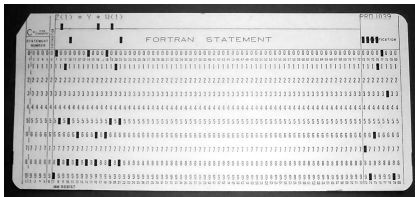
- Late 1960s and 1970s:

  - Hard disks allow for a *direct access* to data

  - Network and hierarchical data models become of widespread use
    - They use graphs and trees as data structures to store data

  - Ted Codd defines the relational data model
    - Would win the ACM Turing Award for that
    - Points of strength: simplicty and possibility of hiding physical details
    - IBM Research begins work on System R prototype
    - UC Berkeley begins work on Ingres prototype
    - They both still exhibit computationally worse performance with respect to network and hierarchical data models
    - OLTP: On-Line Transaction Processing

- 1980s:

  - Relational DB prototypes evolve into commercial systems

  - SQL becomes industrial standard: programmers can now work at the logical level, *without worrying about query optimality and the actual location of data*

  - Parallel and distributed database systems

  - Object-oriented database systems: they use exactly the same model as object-oriented programming languages

- 1990s:

  - Data warehouses: subject oriented, integrated, time-variant, and non-volatile collection of data

  - Decision support systems and data mining applications

  - OLAP: On-Line Analytical Processing

- 2000s:
  - Structured, semi-structured, and unstructured data
  - XML and XQuery: handling of semi-structured data
  - Automated database administration
  - NoSQL systems

- Today:
  - Giant data storage systems (e.g., Google BigTable)
  - Parallel, distributed and cloud systems

- MySQL: https://www.mysql.com/it/

- Oracle: https://www.oracle.com/database/technologies/

- PostgreSQL: https://www.postgresql.org/

- Microsoft SQL Server: https://www.microsoft.com/
  en-us/sql-server/sql-server-downloads