



DMIF, University of Udine

Data Management for Big Data

Relational Databases

Andrea Brunello

andrea.brunello@uniud.it

May 2021



Database Management Systems (DBMSs)

A Database Management System (DBMS) contains information about a particular domain:

- collection of interrelated data
- set of programs to access the data
- safety and concurrency mechanisms, management tools

Databases touch all aspects of our lives:

- Banking: transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases, order tracking



Considering a university database:

- add a new student / instructor / course
- register a student for a course, generate class schedules
- assign grades to students, compute grade averages
- change the salary of an instructor

In the early days, database applications were build on top of file systems:

- ad-hoc solutions: files + dedicated programs
- over time, both grow in their numbers



Drawbacks of Directly Using the File System

Data redundancy and inconsistency:

- multiple file formats (and programming languages)
- same information saved in multiple files

Difficulty in accessing the data:

- need to write a new program to carry out each new task

Integrity problems

- domain constraints (e.g., $\text{mark} \leq 30$) buried in programs
- hard to list/update existing constraints and add new ones



Drawbacks of File System (Cont.)

Atomicity of updates:

- failures may leave the database in an inconsistent state
- example: fund transfer

Concurrent access by multiple users:

- uncontrolled concurrency may lead to inconsistencies
- example: seats booking on an airplane

Security problems:

- hard to manage access rights
- different users may be allowed to access different parts of the data

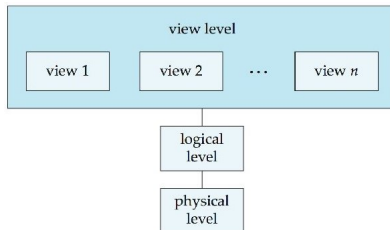


When Should a DBMS be Used?

Characteristics of the data that justify the adoption of a DBMS:

- Large amounts of data: not possible to load all of them into main memory
- Global data: they are relevant for a vast array of users and applications
- Persistence of data: they are there independently from the interacting application or user (unlike program variables)

- *Physical level*: how a record is stored on the disk
- *Logical level*: how data and relationships among them are modeled and represented in the database
- *View level*: different parts of the database shown to different applications / users



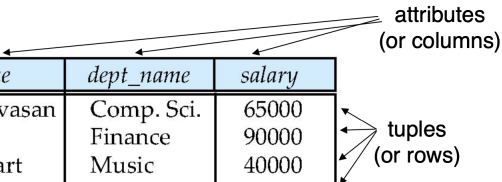


A data model is a collection of tools for describing data and their relationships, semantics, and constraints.

A DMBS may be based on one out of several data models:

- *historical models*: network and hierarchical data models
- *structured data*: relational data model
- *semi-structured / unstructured data*: NoSQL data models

Dominant model because of its simplicity. Collection of tables to represent both data and the relationships among those data.



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



Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are normally required to be **atomic** (indivisible)
- The special value **null** is a member of every domain, and it represents a value that may be:
 - *unknown*: missing mobile phone number of a customer
 - *now know, yet*: a professor just arrived, and still has to be assigned a department
 - *not applicable*: MPG of an electric car



Relation Schema and Instance

- A_1, A_2, \dots, A_n are attributes
- $R = (A_1, A_2, \dots, A_n)$ is a **relation schema**, e.g.,
instructor = (*ID*, *name*, *dept_name*, *salary*)
- Formally, given sets D_1, D_2, \dots, D_n (domains associated to A_1, A_2, \dots, A_n) a **relation** r is a subset of $D_1 \times D_2 \times \dots \times D_n$
- Thus, in our case a relation is a finite set of n -tuples
 (a_1, a_2, \dots, a_n) where each $a_i \in D_i$
- Given our definition, the order of tuples in a relation is irrelevant



Intra-relational constraints:

- not null
- unique
- primary key

Inter-relational constraints:

- foreign key



Primary Keys

- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation r of R
 - $\{ID\}$ and $\{ID, name\}$ are both superkeys of *instructor*
- A superkey K is a **candidate key** if K is minimal
 - $\{ID\}$ is a candidate key for *instructor*
- One of the candidate keys is chosen as the **primary key**
- (Primary) keys must satisfy the *integrity of the entity* condition (not null and unique)

city	hotel_name	hotel_rooms	hotel_stars	hotel_vat_n
Venice	Doge	100	5	64684884645566787
Rome	Ai Fori	300	4	12384884645478721
Milan	Hilton	540	5	56841354324+69534
Milan	Cosmopolitan	235	3	26847854687444455
Naples	Dolce Vita	127	4	75085750925757802
Rome	Dolce Vita	402	3	46848979868746446

Candidate keys:

- $\{hotel_vat_n\}$
- $\{city, hotel_name\}$ assuming that there cannot be two hotels in the same city sharing the same name

Which candidate key should be chosen as the primary one?

⇒ prefer smaller keys



A foreign key is a constraint defined between two relations. It states that the value in one relation must appear in another

- *Referencing* relation
- *Referenced* relation
- Example: *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

The referenced attributes must form a key in their table.

What to do when a referenced value has to be deleted?

- restrict
- delete
- cascade

Example

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

<i>dept_name</i>	<i>building</i>	<i>budget</i>
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



A *transaction* is a unit of program execution that accesses and possibly updates various data items.

A transaction is composed of a set of operations, that can be considered as a single operation from a logical point of view.

For instance, a money transfer from one bank account to another.

Relational DBMSs guarantee some very important properties related to the execution of transactions.



Atomicity All operations in a transaction succeed, or every operation is rolled back (like it didn't happen).

Consistency On transaction completion, the database is moved from a consistent state to a different, but always consistent state (wrt the defined constraints).

Isolation Transactions do not interfere with one another. Contentious access to data is moderated by the database so that transactions appear to run sequentially.

Durability The results of a transaction are permanent, even in the presence of failures of the system. Results can be changed only by a successful, subsequent transaction.



A. Silberschatz, H.F. Korth, S. Sudarshan *Database system concepts*, 7th Edition, 2020.