



DMIF, University of Udine

Data Management for Big Data

Relational model

Andrea Brunello

andrea.brunello@uniud.it

April 2022

The process of designing a database is typically articulated into different phases:

- ① Brainstorming meetings with IT personnel and all interested stakeholders
 - Collection of requirements
 - Design of a conceptual model (e.g., E-R model)
 - The E-R model has a specific notation, the *E-R diagram*, that helps all involved parts to discuss about the future database
- ② Translation of the conceptual model into a logical model
 - Typically, a set of translation rules is followed
 - At this stage, a specific DBMS technology must be chosen (e.g., relational DB)
- ③ Addition to the logical model of details regarding the physical, low-level aspects (e.g., usage of indexes)
 - The physical schema is thus obtained



- We have just seen how to develop the conceptual schema of a database
- The next step is that of choosing a suitable data model to represent the information at the logical level
- Then, the conceptual model gets translated in such a logical model, typically following a specific set of rules
- In this set of slides, we will present the **relational model**

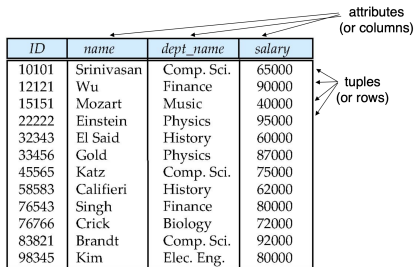


The relational model

Preliminaries

- The **relational model** is a data model that represents information by means of **records** (tuples), that are grouped into **tables** (relations)
- Since 1970s, it has been playing a prominent role in the database realm, because of its simplicity and elegant formalization
- It is based on the mathematical concept of **relation**
- A database organized in terms of the relational model is referred to as a relational database
- Relational databases come with SQL (Structured Query Language), a declarative language that allows for an intuitive interaction with the database and its content

- In the relational model, data are represented by means of a collection of tables (relations), each identified by a name
- All rows (tuples) belonging to the same table are characterized by the same fields (record-based model)
- It is a “rigid” layout, that works best for structured data



ID	name	dept_name	salary
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

Figure: The relation “instructor”



Tables, attributes, and relationships

- By means of attributes, it is also possible to specify logical “links” between tables

<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



Attribute types

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are required to be **atomic**, that is, indivisible (first normal form property)
- This means that we cannot have, in the records, composite fields such as: *address*, made of *city*, *street*, *number*
- Thus, we have two choices:
 - Join together the composing fields into a single attribute
 - Store only the composing fields, each as a separate attribute
- Also, there cannot be multivalued attributes: they should be represented by a table on their own



Attribute types

The *null* value

- The special value **null** is a member of every domain
- It indicates an “unknown” value
- A null value can represent:
 - An unknown information
 - Missing e-mail address of a customer: we don't know if he/she actually has an e-mail account or not
 - An information which is not known, yet
 - A professor may have just arrived and he/she hasn't been assigned a department, yet
 - An information which is not applicable
 - An electric car does not have any meaningful value for the field MPG (Miles Per Gallon)
- As we shall see, null values cause complications in the definition of many operations (e.g., comparisons), thus they should be reduced to the minimal possible amount through sensible schema design choices



Relation schema and relation instance

Example

- A relation instance is composed of a *set* of rows (= no duplicates, order not important)
- Thus, these two relation instances of the same relation schema *instructor* are considered to be as the same

ID	name	dept_name	salary
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

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



Database schema and database instance

- In the same way as we have relation schemas and instances, we may also distinguish between database a database schema and its instance
- A **database schema** is given by its logical design, i.e., it is a collection of relation schemas
- A **database instance** can be seen as a snapshot of the data contained in the database at a given time instant



Integrity constraints

- In order to preserve the consistency of information stored in the database, it is possible (and highly suggested) to define integrity constraints limiting the data that can be stored within the tables
- Intra-relational constraints:
 - Not-null constraint
 - Uniqueness constraint
 - Primary key
- Inter-relational constraints:
 - Foreign key



Not-null constraint

- Defined over a column, it forbids null values
- In the following example, null values are allowed on column *state*
- If we want to model a situation in which every customer is always associated to a proper *state*, then it is possible to define a not-null constraint over the associated column
- As a result, rows with null values for *state* will be rejected by the DBMS

	customername	state	country
▶	Australian Collectors, Co.	Victoria	Australia
	Anna's Decorations, Ltd	NSW	Australia
	Souvenirs And Things Co.	NSW	Australia
	Australian Gift Network, Co	Queensland	Australia
	Australian Collectables, Ltd	Victoria	Australia
	Salzburg Collectables	NULL	Austria
	Mini Auto Werke	NULL	Austria
	Petit Auto	NULL	Belgium

- The uniqueness constraint can be defined over one or more columns
- It forces the values stored in a column or group of columns to be unique among the table rows
- E.g., in a table storing information on hotels, we may forbid the presence of multiple hotels having the same name that are in the same city
- To do that, we can define a uniqueness constraint over the column group $\{Nome_albergo, Citta'\}$

Partiva IVA	Nome albergo	Città	Stelle	Numero camere
1053362646	Ritz	Roma	5	205
1053362600	Ritz	Milano	4	215
1233362688	Da Mimmo	Napoli	3	80
1053369902	Friuli	Udine	3	50
1325239931	Friuli	Udine	2	45



Primary key constraint

- Some attributes play a fundamental role
- Knowing their value, it is possible to uniquely identify a tuple inside a table
- For instance:
 - We can expect *SSN* to uniquely identify a row in a table such as *Person(SSN, Name, Surname, Nationality)*
 - We can expect *VAT_code* to uniquely identify a row in a table such as *Hotel(VAT_code, Name, Stars, Address)*



Primary key constraint

Another example

- Consider the following table, that records information regarding drugs
- We assume each drug to be uniquely identified by its *Code*
- Also, we assume that there may not be two different drugs with same *Name* and *Producer*
- Thus, there are two ways by which we can uniquely identify a row in the table

Code	Producer	Name	Posology
AX124	Bayer	Aspirin	A
AX127	Menarini	VIVIN C	A
AB123	Angelini	Moment 200	B
AB234	Angelini	Tachipirina 500	B
KL253	SANOFI	Enterogermina	C

Figure: Relation “Drug”



Primary key constraint

Definition

- Let $K \subseteq R$ (K is a subset of the attributes of rel. schema R)
- K is a **superkey** of R if the values for K are sufficient to identify a unique tuple of each possible relation $r \in R$
 - E.g., $\{Code\}$ and $\{Code, Posology\}$ are both superkeys of relation *Drug*
- Superkey K is a **candidate key** if K is minimal
 - Minimal: if we remove an attribute from K , then it is not a superkey anymore
 - E.g., $\{Code\}$, as well as $\{Name, Producer\}$ is a candidate key for *Drug*
- One of the candidate keys is chosen as the **primary key**
 - Typically the smallest one



Primary key constraint

Notes

- The primary key plays a fundamental role in relations; if well defined, it prevents the presence of duplicate and possibly inconsistent data
- The primary key constraint naturally implies the presence of two further constraints:
 - **Entity integrity:** attributes belonging to the primary key cannot be null
 - **Uniqueness:** since by definition in a table there cannot be two different rows having the same values for the primary key attributes
- Observe how the candidate keys strictly depend on domain knowledge
 - E.g., if our domain knowledge stated that $\{Name, Producer\}$ are not enough to identify a drug, then it would not have been considered a superkey of relation *Drugs*



Foreign key constraint

- A foreign key constraint is defined between two relations, and it allows us to “link” them
- Intuitively, it states that the values appearing in one relation (**referencing relation**) must also appear in the other relation (**referenced relation**)
 - E.g., given *instructor*(*ID*, *name*, *dept_name*, *salary*) and *department*(*name*, *building*, *budget*) we can define the foreign key $\{instructor.dept_name\} \rightarrow \{department.name\}$
 - Then, every value in *instructor.dept_name* must be present in *department.name* (or be null, if allowed to)
- A foreign key can be composed of multiple attributes
 - *director*(*SSN*, *hotel_name*, *hotel_city*, *salary*)
 - *hotel*(*name*, *city*, *stars*)
 - FK: $\{director.hotel_name, director.hotel_city\} \rightarrow \{hotel.name, hotel.city\}$



Foreign key constraint

Notes

- The attributes referenced by the foreign key must form a superkey in their relation (typically, the primary key)
 - This is quite natural, for otherwise it would not be clear which row we are referring to through the foreign key
- The number and domain of the attributes must be the same in the two relations
- A foreign key can also be defined over the same table, e.g.:
 - *employee*(SSN, name, surname, salary, supervisor)
 - FK: {*employee.supervisor* \rightarrow *employee.SSN*}



Foreign key constraint

Handling row deletions in the referenced relation

- What should be done when a referenced value is deleted?
- Several alternatives:
 - Refuse deletion
 - Delete cascade
 - Set a default/null value

<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



Example

What is wrong with this table?

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

Assume that the primary key is given by {*ID*}



Another example

What is wrong with this table? (Cont.)

- 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