



DIUM, University of Udine

Basi di dati

Relational model

Andrea Brunello

andrea.brunello@uniud.it

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 into such a logical model, typically following a specific set of rules
- In this set of slides, we will start presenting 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), proposed by E. F. Codd in 1970
- Due to its groundbreaking nature, it was only in 1981 that the first relational databases appeared on the market
- It then started to gain popularity and, since the mid 1980s, it has been playing a prominent role in the database realm, because of its simplicity and elegant formalization
- A database organized in terms of the relational model is referred to as a relational database

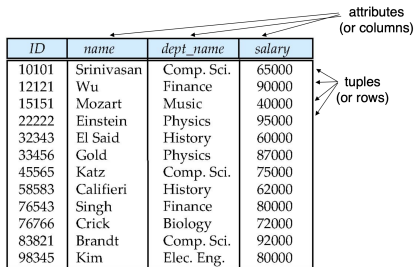


The relational model

Keys of success

- Although the model describes the data very intuitively through the use of **tables**, the latter have a mathematical counterpart, i.e., the concept of **relation** (not to be confused with the concept of relationship in E-R), which provides a sound theoretical basis
- The relational model realizes the *physical data independence*: to access the data, users only have to know its modeling at the relational level; older hierarchical and network models, instead, required them to know details regarding their physical arrangement (ordering on the disk, pointers, ...)
- 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 (links are based on values)
- Differently from previous models which relied on *pointers*

<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 (e.g., string or integer)
- 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:
 - A total lack of information
 - Missing e-mail address of a customer: we don’t know his/her address, but we also don’t know if he/she actually has an e-mail account or not
 - An information which exists but is currently not known
 - A professor may have just arrived at the University and we haven’t been informed about his office location, yet
 - An information which is not applicable
 - An electric car does not have any meaningful value for the field MPG (Miles Per Gallon)



Attribute types

The *null* value – caveats

- Null values are useful because they allow us not to use fictitious values to represent unknown information (e.g., “-1” for an attribute *quantity*), that could pose several subtle issues
- Nevertheless, they 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

- Formally, in mathematics, given sets S_1, S_2, \dots, S_n , a **relation** is a subset of $S_1 \times S_2 \times \dots \times S_n$
- Let A_1, A_2, \dots, A_n be **attributes**, each A_i with domain D_{A_i}
- $R(A_1, A_2, \dots, A_n)$ is a **relation schema**
 - E.g., $\text{instructor}(\text{ID}, \text{name}, \text{dept_name}, \text{salary})$
 - It can be seen as the skeleton of a table
- A **relation instance** r is a subset of $D_{A_1} \times D_{A_2} \times \dots \times D_{A_n}$
 - E.g., $r \subset D_{\text{ID}} \times D_{\text{name}} \times D_{\text{dept_name}} \times D_{\text{salary}}$
 - It can be seen as the set of rows of a table
 - Observe that, as a result, table rows are *unordered*
- Unlike mathematical relations, relation instances are *finite*
- Notation:
 - $r \in R$: relation instance r of relation schema R
 - $t \in r$: tuple t of relation instance r (row in a table)



Relation schema and relation instance

Example

- Since the order of tuples is irrelevant, these two relation instances of the same relation schema *instructor* are considered to be as the same
- Note that there cannot be, by definition, repeated rows

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



- In the same way as we have relation schemas and instances, we may also distinguish between 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
 - General tuple-level constraints
- 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*
- To model a situation in which every customer is always associated to a proper *state*, 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 (note how this changes the semantics of the modeled domain)

	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



Uniqueness constraint

- 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 $\{Nome_albergo, Citta'\}$
- Note how this is different than defining two uniqueness constraints, over $\{Nome_albergo\}$ and over $\{Citta'\}$

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, City)*
 - Note how, in the same table *Hotel*, the pair (*Name, City*) may or may not uniquely identify a hotel; it really depends on the kind of reality that we want to model



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, with respect to the considered domain, 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



Primary key constraint

Notes

- Observe how the definition of candidate keys strictly depends on domain knowledge
- For instance, in the following table, even if there are no two students with the same name and surname, it is not sensible to consider such two attributes as a candidate key
- The two “natural” candidate keys are *sid* and *login*; thus, one of them will be set a primary key constraint, and the other one a uniqueness constraint

sid	fname	lname	login	age	GPA
C123456	John	Smith	smithx02@ece.abc.edu	19	3.3
C234561	Karen	Johns	johnsx05@cs.abc.edu	18	3.0
C345612	Mary	Anderson	anderx10@math.abc.edu	20	3.5
C456123	Helen	Henderson	hendex05@ece.abc.edu	18	2.9
C561234	John	Smith	smithx99@cs.abc.edu	19	3.8
C612345	Aidan	Cocke	cockex35@ece.abc.edu	18	3.3



- A natural candidate key of relation *Book*(*ISBN*, *author*, *year*, *editor*, *genre*, *num_pages*) is ISBN
 - And if we further assume that an author can write at most one book in a given year?
 - And if we further assume that an author can write at most one book in a given year for a given editor?
- Notation: we can specify the chosen primary key of a relation by underlining its attributes, for instance, *Book*(*ISBN*, *author*, *year*, *editor*, *genre*, *num_pages*)



General tuple-level constraints

- Sometimes, it is useful to define more general constraints at the tuple-level, i.e., some criteria that have to be satisfied by a tuple as a whole
- For instance, in a relation *Exam(student, course, mark)*, we could specify that the mark should be between 0 and 30
- Such constraints can be defined by means of suitable **boolean expressions**, that make use of connectives such as *AND, OR, NOT*
- Relational databases provide specific constructs to specify general constraints at the tuple-level. In Postgres, we have the *CHECK* clause



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: $\{director.hotel_name, director.hotel_city\} \rightarrow \{hotel.name, hotel.city\}$



- 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

Graphical example

Department

EmpNo	EmpName	DepNo
1001	Sahil	101
1004	Kavish	102
1006	Aditya	103
1005	Atul	104

Foreign Key

Relationship

Employee

Primary Key

DepNo	DName	Location
101	HR	Delhi
102	Sales	Bangalore
103	Marketing Executive	Hyderabad
104	Technical Engineer	Chennai



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

Let us consider the following relational schema about authors and books

AUTHOR(name, surname, birth_date, nationality)

BOOK(title, genre, num_pages, publication_year)

WRITES(author_name, author_surname, book_title, book_year)

- Let us assume each author to be uniquely identified by his/her name and surname, and further described by his/her birth date and nationality.
- We also assume that there may be more than one book with the same title, however, given a year, that cannot be two different books having the same title.



Exercise 1

- Finally, consider that a book may be written by more than one author, and an author can write, in general, more than one book.

Define the primary keys, uniqueness constraints (if any), and foreign keys (if any). Possibly justify your choices by providing some assumptions.

Let us consider the following relational schema about speakers and conferences

SPEAKER(*name*, *address*, *telephone*, *email*)

ATTENDANCE(*speaker_name*, *conference_name*, *conference_year*)

CONFERENCE_EDITION(*name*, *year*, *city*)

- Let us assume each speaker to be uniquely identified by his/her name, and further characterized by his/her address, phone number and personal email.
- Assume that each edition of a conference is uniquely identified by the name of the conference and the year in which the specific edition was held (for example, ICDM 2019 and ICDM 2020 identify two different editions of the ICDM conference).

- Also, consider that a given conference edition (e.g., ICDM 2024) cannot be hosted by multiple cities.
- Finally, let us assume that at each conference edition there can be several speakers and that the same speaker can participate, in general, to multiple conference editions.

Define the primary keys, uniqueness constraints (if any), and foreign keys (if any). Possibly justify your choices by providing some assumptions.

Let us consider the following relational schema about musicians and bands.

MUSICIAN(name, surname, art_name, nationality, band)

BAND(name, foundation_year, genre, frontman)

- Let us assume each musician to be uniquely identified by his/her art name.
- Also, assume each band to be uniquely identified by its name.



Exercise 3

- A musician can be part of at most one band; and, a band can be composed of multiple musicians.
- Finally, each band has a frontman, who is a musician himself/herself.

Define the primary keys, uniqueness constraints (if any), and foreign keys (if any). Possibly justify your choices by providing some assumptions.