



University of Udine

Data Management for Big Data

The relational model

Andrea Brunello

andrea.brunello@uniud.it



The relational model

- 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

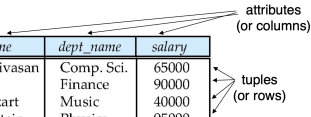
- Although the model describes the data very intuitively through the use of **tables**, the latter have a mathematical counterpart, i.e., the concept of **relation**, which provides a sound theoretical basis
- The relational model realizes the *physical data independence*: to access the data, users only 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



The relational model

Representation of information

- 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)

<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

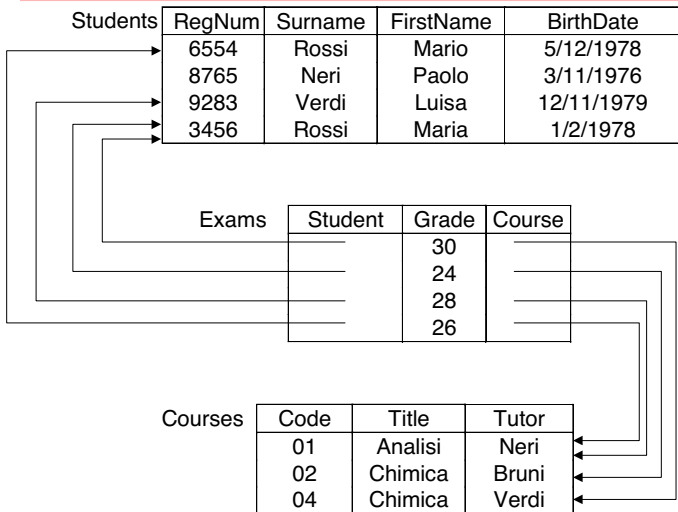
<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Students	RegNum	Surname	FirstName	BirthDate
	6554	Rossi	Mario	5/12/1978
	8765	Neri	Paolo	3/11/1976
	9283	Verdi	Luisa	12/11/1979
	3456	Rossi	Maria	1/2/1978

Exams	Student	Grade	Course
	3456	30	04
	3456	24	02
	9283	28	01
	6554	26	01

Courses	Code	Title	Tutor
	01	Analisi	Neri
	02	Chimica	Bruni
	04	Chimica	Verdi





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



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



Integrity constraints

- In order to preserve the consistency of data stored in the database, it is possible (and highly suggested) to define integrity constraints, that limit 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 want to 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 $\{Nome_albergo, Citta'\}$
- What about defining two uniqueness constraints, the first over $\{Nome_albergo\}$ and the second 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 relation R)
- K is a **superkey** of R if the values for K are sufficient to identify a unique tuple within $r \in R$
 - 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
- What about the other keys?
 - They can be encoded e.g. by means of unique constraints



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 allows one to identify the data contained in a table, and it is also through primary key values that the relationships between tables are built
- 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



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 is defined between two relations, and it allows to “link” them, establishing a *one-to-many* relationship between the tuples
- 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

Graphical example

Employee

<u>Code</u>	Name	Department
0101	Mary	R&D
0105	Frank	Sales
0102	John	Marketing
0110	Juliet	Sales
0109	Violet	Sales

Foreign key



Department

<u>Name</u>	Location	Budget
R&D	Venice	R&D
Sales	Venice	Sales
Marketing	Rome	Marketing
Product	Milan	Sales



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*}

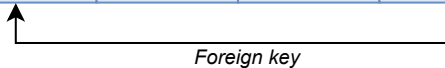


Foreign key constraint

Example: FK over the same relation

Employee

<u>Code</u>	Name	Department	Supervisor
0101	Mary	R&D	0105
0105	Frank	Sales	
0102	John	Marketing	0105
0110	Juliet	Sales	
0109	Violet	Sales	0110





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

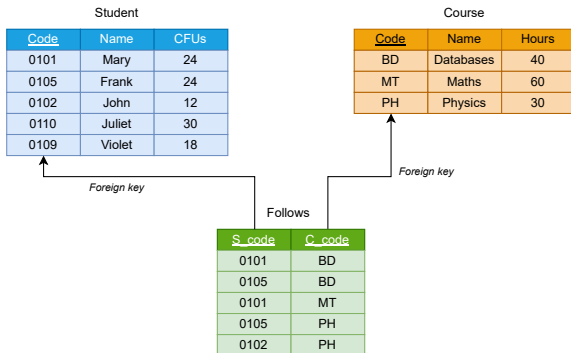
(a) The *instructor* table

<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

Many-to-many relationships

- We have seen scenarios in which foreign keys are used to encode *one-to-many* relationships between tables
- What about *many-to-many* relationships?



What is wrong with this table (primary key is given by *ID*)?

<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



Caveat (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
- In addition, **conceptual modeling** techniques should be used to design a database in a sound manner



Atomicity All operations in a transaction succeed, or every operation is rolled back (like nothing happened).

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 subsequent successful transaction.



Relational DBMS main vendors

- 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>

