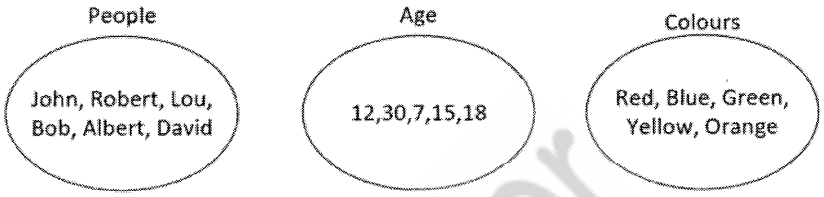
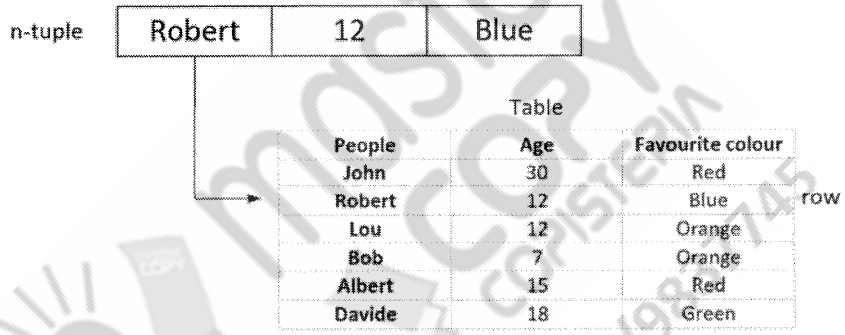
Relational Model

26 September 2017 16:15

Structure of Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name. In general, a row in a table represents a **relationship** among a set of values. Since a table is a collection of relationships, there is a correspondence between the concept of table and the mathematical concept of **relation** (from which comes the name *relational database*).
In mathematical terminology a **tuple** is a sequence of values, thus a relationship between n values can represented by an **n-tuple** of those values and the latter corresponds to a row in a table.





People	Age	Favourite colour
John	30	Red
Robert	12	Blue
Lou	12	Orange
Bob	7	Orange
Albert	15	Red
Davide	18	Green

MASTER COPY
Tel. 050 831 21 63
Cell. 388 9857745

In a relational database model we can define:

- Relation: a table
- Tuple: a row of the table
- Attribute: a single column of a table

Since tuples are obtained by unordered sets, as well the order in which row appears in table is not relevant from the point of view of the relation itself.
For each attribute of a relation a domain of permitted values must be established. The latter must be atomic (the element of the domain must be indivisible) and can contain also the **null** value, which means unknown.

Database Schema

We must differentiate between **database schema**, which is the logical representation of the database, and the **database instance**, which is a snapshot of data contained in a database for a given instant of time. The **relation schema** is a list of attributes and their corresponding domain, capable to completely describe a table.
Thus, given a table, its content (the instance) may change over time, instead the relation which it represents remains constant over time.
Considering a table which describes all the departments in a university, together with the building where each of them is located and the related budget, a possible instance could be:

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

Which is a table completely described by the following relation schema

department(dept_name, building, budget)

Considering now to escribe all the set of instructors of the academy, each of them identified by a unique ID, name, the name of the associated department and the salary payed by the university. The relation schema is:

instructor(ID, name, dept_name, salary)

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

MASTER COPY
Tel. 050 8312126
Cell. 338 9837745

Keys

Tuples are distinguished according their attributes, which means that cannot exist two different tuples with all the same attributes. There must be a such way by tuples can be uniquely identified and the latter is achieved by the concept of **super key**. Hence a super key is a set of one or multiple attributes which taken collectively allow us to uniquely identify a tuple in a given relation.
Formally, denoted R a set of attributes for the relation r, if we say that a subset K of R is a superkey for r, then must be that:

if t_1 and t_2 are touples in r and $t_1 \neq t_2$, then $t_1.K \neq t_2.K$

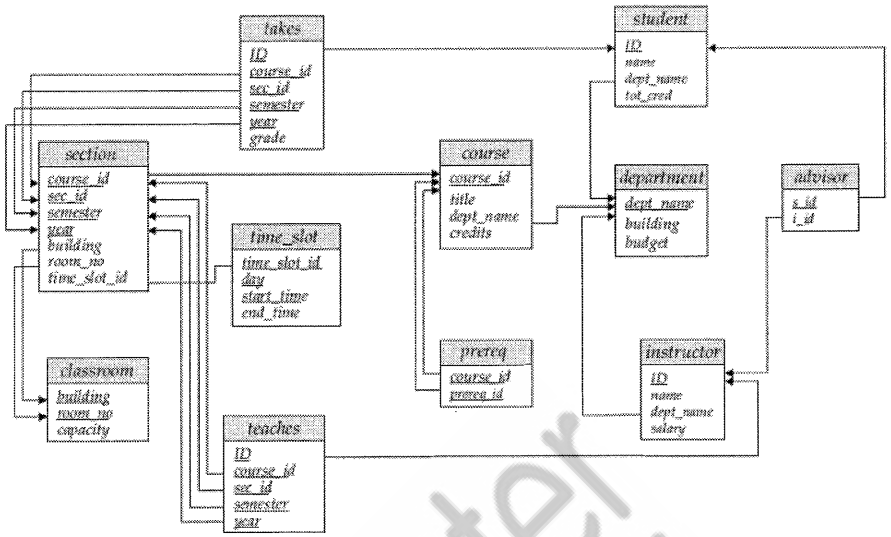
An example of superkey could be in the instructor table the ID attribute, which is unique for each row (for each tuple). If K is a superkey, then any superset of K will be as well: for instance, a superset of K in instruction table could be the couple of attributes (ID,name), cause uniquely identifies each instructor in the table. The concept of superset is needed, otherwise we could not have a superkey. If, in fact, we only consider the name attributes, this cannot be a superkey because may exists two instructors with the same name.
We are often interested in some particular superkeys for which no proper subset is a superkey, the latter called **candidate keys**. In other words, candidate keys are set of attributes which forms a superkey, but none subset of them is a superkey alone.
For instance, suppose that the couple (name, dept_name) is sufficient to uniquely identify a person, then, (name, dept_name) is a candidate key. If we consider only dept_name or name, the latter attributes are not sufficient for a unique identification of a tuple.
From candidate keys it is possible to derive the concept of **primary key**, which is a candidate key, chosen by database designer, with the main purpose of distinguish tuples in a unique way within a relation.
It is customary that primary key is listed as first attribute in a table schema, thus *dept_name* and *ID* are primary keys respectively for tables *departments* and *instructors*.
May happen also that a relation r_1 include among its attributes a primary key for relation r_2. In this situation we call the attribute **foreign key**, r_1**referecing relation** and r_2 **referenced relation.**
For instance, in the *instructors* table, the attribute *dept_name* is a foreign key referencing the table *departments*, where the same attribute is the primary key.
The foreign key, if exist, must be for all tuples: this means that cannot exist a tuple in instructors table with a value of dept_name which is not present in the department table. Formally, in a relational database, given a tuple from instructor table $t_{instructor}$, ther must exist some tuple in departments table, $t_{departments}$ such that

$$t_{instructors}.dept_name = t_{departments}.dept_name$$

Once each relation has been defined through a schema, it is possible to realize the schema diagram, which is composed by the following components:

- Each relation (table) is represented by a square

- The name of the relation is boxed alone
- The first attribute is the primary key for that relation
- A single arrow represent a foreign key: starts from the foreign key in the referencing relation and points to the primary key of the referenced relation.



Relational Operation

Procedural relational query languages provides a set of operation that can be applied on a single relation or a pair of ones. These operations return always a single relation, thus complex operation can be performed simply applying them in sequence, one on the result of the previous.
The most frequent operation is the **selection** of specific tuples from one or more relation, according to a predicate: select all the instructor with a salary larger than 85000$. The operation is applied on instructors 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

MASTER COPY
Tel. 050 8312126
Cell. 388 9837745

and provide the following result

ID	name	dept_name	salary
12121	Wu	Finance	90000
22222	Einstein	Physics	95000
33456	Gold	Physics	87000
83821	Brandt	Comp. Sci.	92000

Another frequent operation is the selection of only some attributes of the tuple: in this case, the result will be again a relation with a reduced number of attributes with respect to original one, in particular only the requested attributes.
The **join** operation allows the combining of two relations by performing the merge of pairs of tuples, one from each relation, into a single tuple. There are several ways to perform join operations, one of the most common is called **natural join** (⋈) which combines pairs of rows from the two input realtions that have the same value on al attributes that have the same name. For instance consider the table of the instructor and the department one, the result of natural join will be a new table with a record for every time that *instructor.dept_name = departments.dept_name*.

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

The result will contains all the attributes of both relations, if not differently specified, which is performing a select operation on the join operation result.
Another operation is the cartesian product, which provides as result tuples from both relation regardless any attributes with the same name.
Since tables are set, also the most common operations on set are allowed, as union and intersection.

Symbol	Name	Example	Description
σ	Selection	$\sigma_{salary \geq 85000}(instructor)$	Select all the attributes of the instructors with a salary higher or equal to the threshold
Π	Projection	$\Pi_{ID,salary}(instructor)$	Select attributes ID and salary from instructor relation
$\bowtie$	Natural join	$instructor \bowtie department$	Output a pair of row, one from instructor, one from department, when an attribute from instructor matches an attribute with the same name from department
X	Cartesian prod	$instructor\ X\ department$	Output all pairs with all possible combination of tuples
$\cup$	Union	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$	Output all the attribute name from instructor and all the ones from student.

As said, since the result of an operation is still a relation, more than one operation can be applied in sequence, in order to be able to perform even more complex operation: suppose that we want to select the attributes name, salary and the department of all the instructors with a salary higher than a threshold:

1) perform a join operation and obtain a relation with instructors and associated departments
2) perform a select operation on the obtained relation according to the predicate obtaining a new relation
3) perform again a selection with attributes specified.

MASTER COPY
Tel. 050 8312126
Cell. 388 9637745

Introduction to SQL

26 September 2017 19:48

An SQL query con contains three types of clauses:

- **select**: list the desired attributes in the relation result
- **from**: list of relation to be accessed by the query
- **where**: predicate based on attributes of the relations lister in the from clause.

A general SQL query can be defined as:

select $A_1, A_2, \dots, A_n$
from $r_1, r_2, \dots, r_n$
where P

Where A_i is an attribute, r_i is a relation and P the predicate which, if omitted, is **true.**
Performing a query on multiple relations, such as:

select *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID* ***and*** *instructor.dept_name = 'Comp.Sci';*

MASTER COPY
Tel. 050 8312126
Cell. 388 9837745

Involves into the following steps:

1) Generate a cartesian product of the relation listed in the from clause
    a. This usually involves into a relation composed by a huge number of tuples
2) Apply the predicate on where clause to result obtained in step 1
    a. This will reduce a lot the number of tuples in the result realation
3) For each tuple as result of step 2, output the listed attributes

Of course, this is only a way to understand how things work but it is not real way in which are implemented: first of all for the cartesian product operation, which involves into a huge number of tuple, which the larger part will be discarded by the predicate application. For this reason, queries are optimized as far as possible in order that the cartesian product just satisfies the where clause.

Natural Join

The natural join operation operates on two relations producing another relation as result. The latter will be a set of tuples every time that an attribute in the first relation has the same value of another attribute, with the same name, in the second relation.
Thus it is equivalent to write:

Select *name,course_id*
From *instructor, teaches*
Where *instructor.ID = teaches.ID;*

Select *name,course_id*
From *instructor* ***natural join*** *teaches;*

String Operation

SQL expresses patterns by using **like** operator:

- 'Intro%' matches any string beginning with "intro"
- '%Comp%' matches any string which contains "Comp"
- '---' matches any string of exaclty three characters
- '---%' matches any string with at least three characters

Select *detp_name*
From *department*
Where *building* ***like*** *'%CAS%'*