**Chapter 2: Introduction to the Relational Model**

Table of Contents

[Database Schema 3](#_Toc49497437)

[Keys 4](#_Toc49497438)

[Foreign Keys 5](#_Toc49497439)

[On Delete Cascade 6](#_Toc49497440)

[Selection 7](#_Toc49497441)

[Cartesian Products 8](#_Toc49497442)

[Natural Join 9](#_Toc49497443)

A relational database consists of a collection of tables, with each table being given a unique name. For example, this is the instructor table:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | name | dept\_name | salary |
|  |  |  |  |
|  |  |  |  |

Each instructor in the instructor table can be identified using the ID. A row in this table can be thought to represent the relationship between a specific ID, and the corresponding name, dept\_name and salary. In mathematics, a ***tuple*** is simply a list of values. Thus, in the relational model, ***relation*** is used to refer to a table, and ***tuple*** is used to refer to a row. Similarly, ***attribute*** is used to refer to a column. The above table has four attributes.

The term ***relational instance*** is used to refer to a specific instance of a relation, i.e. the specific set of rows in a table at some point of time. The order in which those tuples appear is irrelevant.

Each attribute of a relation has a set of permitted values, known as its ***domain***. For example, the domain of the name attribute should be strings. Each domain is required to be ***atomic***, or indivisible. For example, a set of phone numbers is not atomic, but a single phone number is. However, it is important to understand that what a domain contains is not actually important. It is only important how we use it. A single phone number can still be divided into an area code and a country code, making it non-atomic, but as long as we treat the whole phone number as a single unit, it is considered atomic.

The ***null*** value is used to signify that a value is unknown or does not exist. null values cause a significant number of problems and should be eliminated if possible.

## Database Schema

A relation is like a variable in the programming language, while a ***relation schema*** is like a type definition. A relation schema contains the list of attributes and their corresponding domains, but it does not contain any data. The contents of a relation may change, but the schema generally does not.

Consider a department relation. The schema is:

department (dept\_name, building, budget)

Notice that dept\_name appears here as well as under instructor. Using common attributes is one way of relating tuples of distinct relations. Suppose we want to find all instructors who work at a particular building. We would first look in the department relation to find the dept\_name of all departments housed under that building, and then look at the instructors relation to find all instructors for that department.

## Keys

We need to be able to distinguish tuples inside a specific relation. This is done in terms of the attributes. Thus, the values of a tuple must be put in such a way that it is uniquely identifiable. No two tuples can have the same values for all attributes.

A ***superkey*** is a set of one or more attributes that collectively allow us to identify a tuple in the relation. For example, the ID attribute in the instructor relation is a superkey. The name attribute however, is not, since it is possible that two instructors have the same name.

A superkey may contain attributes that are irrelevant. The combination ID and name is a superkey for the instructor relation. Every superset of a superkey is also a superkey. A superkey that is minimal and can uniquely identify any tuple on its own, is called a ***candidate*** key. The combination of ID and name here is not a candidate key, but ID alone is. We can also have multiple keys of any sort. For example, phone\_number and ID are both candidate keys.

A candidate key that is chosen by the database designer to serve as the means through which tuples will be identified is called a ***primary key***. A primary key should be chosen so that it is never, or at least very rarely, changed. For example, the address attribute is a poor choice, but the social\_security\_number attribute is a good one. It is customary to list the primary key attributes before other attributes in a schema. The primary key attributes are also underlined.

## Foreign Keys

A single relation may contain repetitive data. For example, a relation storing details of students may contain some detail about their respective departments that are repeated for students from the same department. This will cause two problems. Firstly, repeated data is a waste of space and secondly, if there is a change to the repeated data, the change must be made possibly hundreds of times, which is likely to cause mistakes. This condition is called redundancy.

Another condition that may be faced is a large number of empty values for some particular data. For example, a relation containing information about citizens may contain information about their passports. However, not all citizens have passports, which will result in a large number of null values. Null values cause problems, plus this is just poor design.

The solution to these problems is to have the data that is likely to repeat or can be put into a set in another relation, and ensure that that relation has a primary key. We can then put just the values of the primary key attribute in the original table, linking the two relations. The original relation is called the referencing relation, and the new relation is called the referenced relation. The primary key being put in the referencing relation is called a foreign key for that relation.

Using foreign keys is important for two reasons. Firstly, it removes redundant data. Secondly, it ensures that, if a value is present, it comes from a valid source. If no value is present of course, a null value is given.

### On Delete Cascade

When declaring a foreign key, we can set it up so that when any record in the referenced table is deleted, the corresponding records in the referencing table are also deleted. This is called a cascade delete.

## Selection

In selection, only tuples that match the condition given are shown. selects tuples matching the condition from the relation .

We can also select columns. selects columns and from the tuple . After the selection we may have repeated data, for example, two tuples that have the same value in column and but had different values in column . Repeated data is shown only once.

## Cartesian Products

The cartesian product of two relations contains all possible combinations of the two relations. For example, if a relation had 2 records and a relation had 4 records, the cartesian product would have 8 records.

Cartesian products contain both valid and invalid data. For example, the cartesian product of a relation that contains information about students and a relation that contains information about departments will give us tuples that combine a student from one department with the department information of his own department and other departments as well. The latter information is worthless.

We have other ways to join records like this, such as unions, where the records are simply added to one another and shown (common records shown only once), difference, where shows records present in but not in , and intersections where only records present in both relations are shown.

## Natural Join

In natural join, the tuples from a relation are checked against the tuples in a relation . If the two relations have some common attribute and the values of those attributes are the same in both tuples, then the tuples are joint and shown. For example, a student relation could have one attribute named dept. while a department relation could have the same attribute. A natural join would thus join all tuples from student with the data for their respective departments from department. If there is more than one attribute common, both attributes must have the same value. No matter how many times the tuples appear in a relation (due to differences in other attributes not being considered), as long as the common attributes are the same in both relations, the tuples will be combined.