Course on Database Design

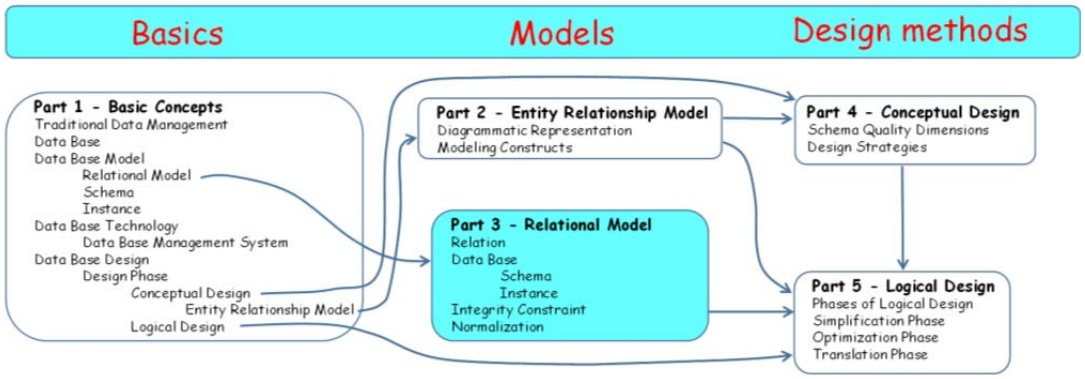
Carlo Batini

University of Milano Bicocca

Part 3 – Relational Model

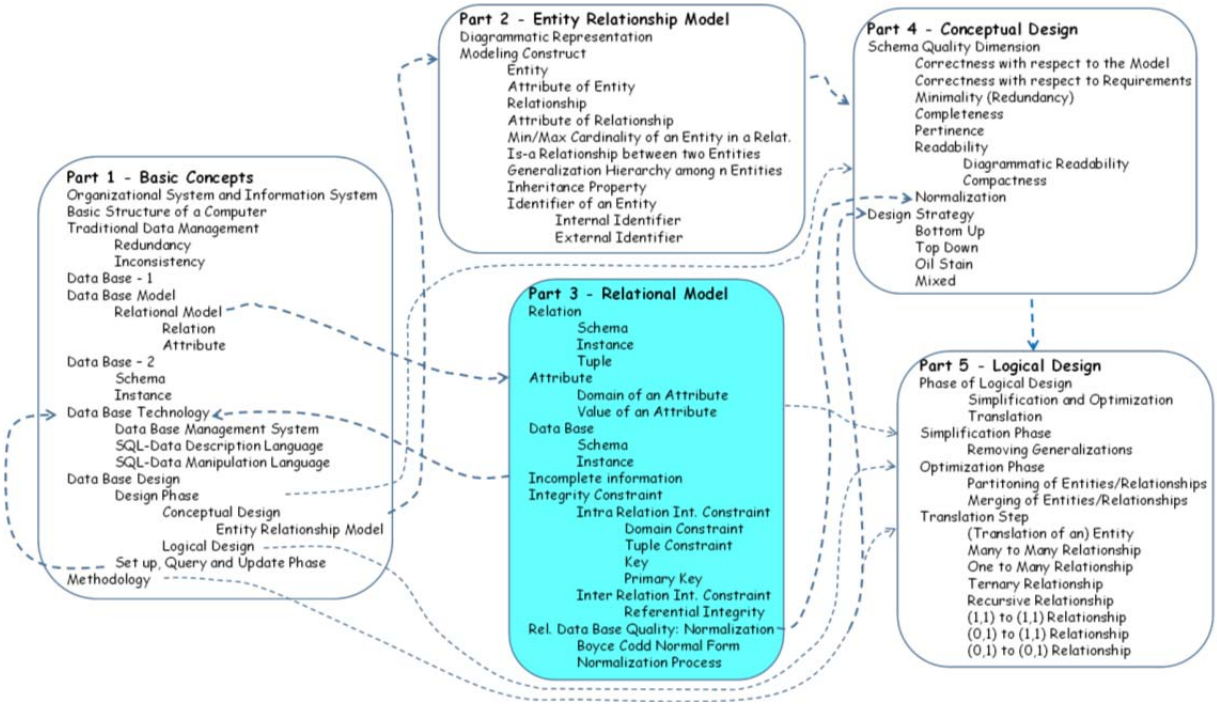
Part 3 – Lesson 1 - Introduction to the Relational model

We face now the second model discussed in the course, the relational model, adopted in most of database management systems used all over the world. So, we have reached the turning point of the course.



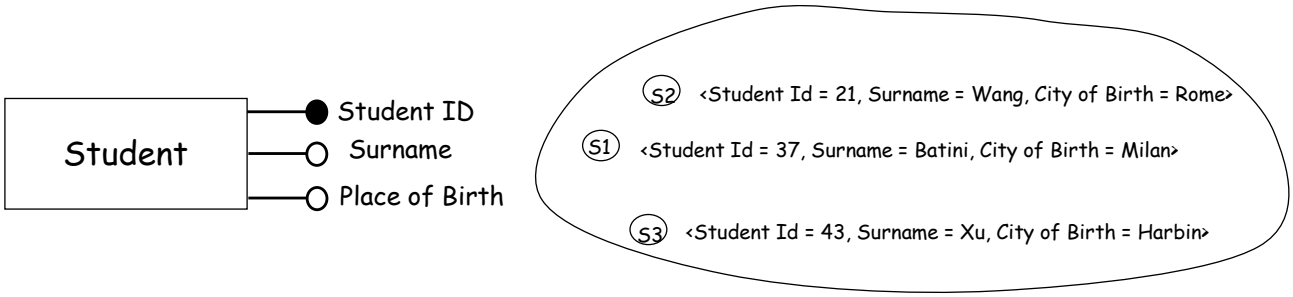
High level conceptual map

This figure shows in more detail the relationships between this third part of the course and the other parts. We will examine in depth some concepts previously discussed in the introduction, and will lay the foundations for the part of the course in which we discuss design methodologies.



Low level conceptual map

Let us start with the usual motivating example. Look at this figure.



An ER schema and its instance

We see an entity, Student, with three attributes, and a corresponding instance made of three student instances, represented with the attributes and related values.

Question 3.1 - Now try to represent the entity and its instance by means of a relation (also called table) that we have introduced in Part 1.

Answer to Question 3.1

Probably you produced the following table, with three columns corresponding to the three attributes of the entity, and three tuples corresponding to the three instances of students.

Student

Student Id	Surname	Place of Birth
21	Wang	Rome
37	Batini	Milan
43	Xu	Harbin

A relation having the same information content of the Entity Relationship schema and instance

Manipulating relations

Now before defining the main concepts of the relational model, I would like to transform the above relation together with you, to discover relevant behaviors of data in the relational model. Let us see two examples of transformations.

Example 3.1 - In the first transformation, we decompose the relation Student into two smaller relations Student1 and City

Student

StudentId	Surname	City of Birth
37	Batini	Rome
41	Rossi	Rome
53	Wang	Harbin

Student1

StudentId	Surname
37	Batini
41	Rossi
52	Wang

City

City of Birth
Rome
Harbin

Splitting a relation into two smaller relations

The two relations have the following attributes. The first relation, Student1, has attributes StudentId and Surname, and the second, City, has a unique attribute City of Birth. Notice that in the second relation we have deleted one occurrence of Rome, because a relation is a mathematical set, and so we may have only one instance of Rome.

This transformation does not seem a “good” transformation. What is a “good” transformation? Ii is a transformation that allows you to reconstruct the original undivided relation. The only way in

which we can try to reconstruct the original transformation corresponds to evaluate the Cartesian product of the first and second relation, but the Cartesian product is made of six tuples, as we have to couple each one of the three tuples of Student1 with each one of the two tuples of City. Not good.

Example 3.2 - In this example we decompose the relation Student into two relations, Student1 and Student2, as before, but in this case we have an attribute in common, that is StudentId. If we try to reconstruct the original relation from the two relations, in this case we succeed. This is reasonable, as StudentId univocally identifies the Surname and the City of Birth. To each StudentId, a unique Surname and a unique City of Birth correspond. We will say soon that StudentId is the *key* of the relations Student, Student1 and Student2.

Student

StudentId	Surname	City of Birth
37	Bqtini	Rome
41	Rossi	Rome
53	Wang	Harbin

Student1

StudentId	Surname
37	Batini
41	Rossi
52	Wang

Student2

StudentId	City of Birth
37	Rome
41	Rome
52	Harbin

An example of “good” transformation

Example 3.3. Let us finally examine a third transformation, in which, again, we decompose the original relation into two relations made, in this case, of two attributes. This time the two pairs of attributes are respectively StudentId and City of Birth and Surname and City of Birth.

Student

StudentId	Surname	City of Birth
37	Batini	Rome
41	Rossi	Rome
53	Wang	Harbin

Student1

StudentId	City of Birth
37	Rome
41	Rome
52	Harbin

Student2

Surname	City of Birth
37	Rome
41	Rome
52	Harbin

Another “not good” transformation

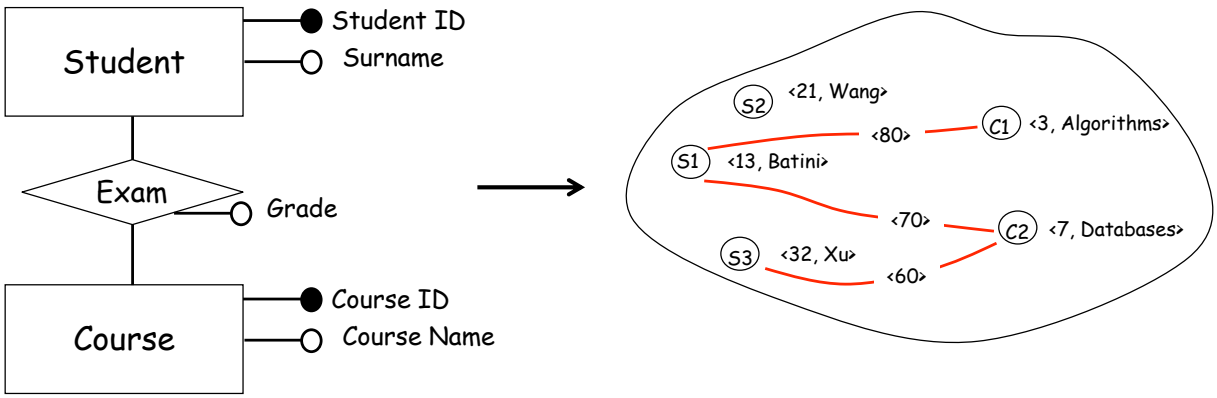
In this case we do not have StudentId as a common attribute of the two tables. As in the first case, we do not have any means to reconstruct the original table; if we use the common attribute City of Birth to link the two tables, we couple the first two tuples of the first table and the first two tuples of the second table, resulting in a table of five tuples instead of three.

In the discussion of the above examples we have seen that, when for some reason we decompose a table into a set of tables, we have to choose carefully the attributes of the tables; according to the type of transformation, we may be able, or else we may not be able, to reconstruct the original information content. In the last lesson of this part, focused on the concept of *normalization*, we will formalize this concept, finding general conditions of what we can call *transformation without loss of information*.

The relational model is based on values

Now we focus on another fundamental characteristic of the relational model.

Question 3.2 - Look at the ER schema in the figure and try to represent the schema and the instance by means of relations. Notice that in the instance we represent values of Grade with tags associated to instances of the relationship Exam.



An ER schema and an instance

I suggest you to represent the ER schema and the instance in the relational model by means of three relations.

Discussion on Question 3.2

You should have produced three relations as the following ones.

Student

Student Id	Surname
21	Wang
13	Batini
32	Xu

Exam

Student Id	Course Id	Grade
13	3	80
13	7	70
32	7	60

Course

Course Id	Name
3	Algorithms
7	Databases

Three relations representing students, exams passed and courses

Question 3.3

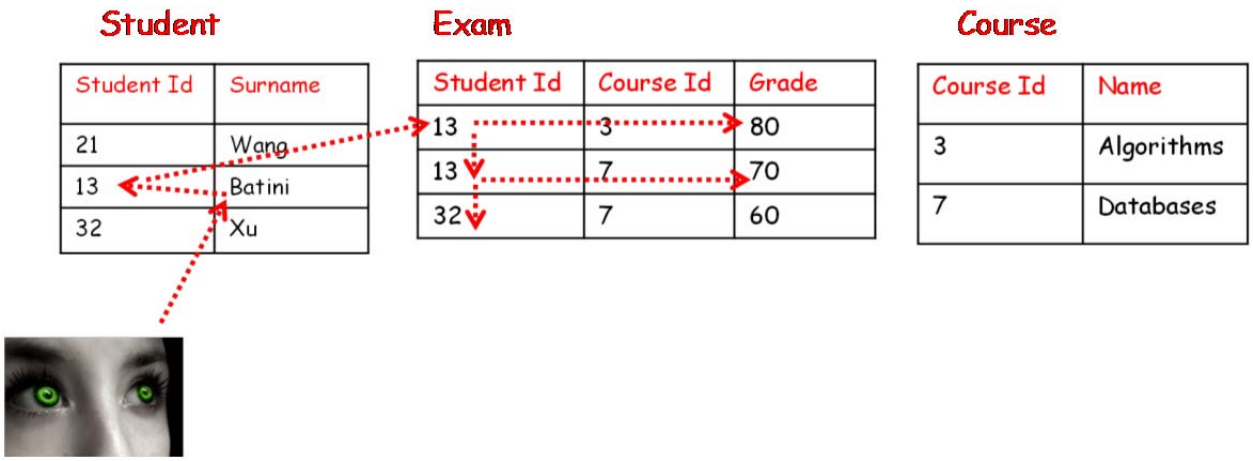
Now, try to reply to the query

Tell me the average grades in the exams passed by the Student with Surname = "Batini"

and try to trace the movements of your eyes, drawing them with a pen.

Answer to Question 3.3

The answer is 75, and it is possible that the movements of your eyes have been as in the following figure.



The relational model is based on values

In order to reply to the question, you had to navigate in the tables, and the navigation between Student and Exam has identified the tuples that have the same value of Batini's Id (i.e. 13). We say that *the relational model is based on values*, meaning that links between pairs of the same object of the real world represented in different tables (in this case, the student Batini), are made possible by the explicit representation of the same values (in this case the values of the Student Id) in different tables. The same characteristic helped us to reconstruct the relation that has been decomposed in two relations in the example 3.2. With this important consideration, we conclude the lesson.

Part 3 – Lesson 2 – Basic definitions of the relational model

We now introduce the basic concepts of the relational model, namely:

1. Relation schema
2. Database schema
3. Attribute
4. Tuple
5. Relation (or Table) instance
6. Database instance

Let us consider again the relations of the previous example, related to a University database.

Student

Student Id	Surname
21	Wang
13	Batini
32	Xu

Exam

Student Id	Course Id	Grade
13	3	80
13	7	70
32	7	60

Course

Course Id	Name
3	Algorithms
7	Databases

A database made of three relations

Definition - The schema of a relation R (or relation schema) is the set of concepts represented in the relation; it is made of the relation name R and its properties, also called *attributes*. In the example we have three relation schemas, corresponding to

1. Student, with attributes Student Id and Surname,
2. Exam, with attributes StudentId, CourseId and Grade, and
3. Course, with attributes CourseId and Name.

Notice that we can represent, e.g. the relation schema associated to Student as

Student (StudentId, Surname)

Definition - The schema of the data base (or database schema) is the set of relation schemas of the database.

In the example we have a unique database schema University made of

Student (StudentId, Surname)
Exam (StudentId, CourseId, Grade)
Course (CourseId, Name)

Notice that the values represented in relations *are not part of the schema*.

Definition - Given a relation schema R, an attribute is defined as a pair <Attribute Name, Domain> where Attribute Name is the name of the attribute and domain is the set of values that the attribute may have in the relation instance.

E,g, if Students in the University we are describing are 20.000, the attribute StudentId can be defined as

< StudentId, [1..20.000]>

Definition - Given a relation schema R (A1, A2, …, An), a tuple is defined as <A1: v1; A2: v2; …; An, vn> where for each Ai vi is a value in the domain of Ai.

E.g. in the relation schema Student (StudentId, Surname, Place of Birth) a tuple is

<Student Id: 21; Surname: Wang; Place of Birth: Rome>

Definition - An instance of a relation R (or relation instance) is a set of tuples defined on the attributes in R.

In our example an instance of the relation Student is

Student

StudentId	Surname	Place of Birth
21	Wang	Rome
37	Batini	Milan
43	Xu	Harbin

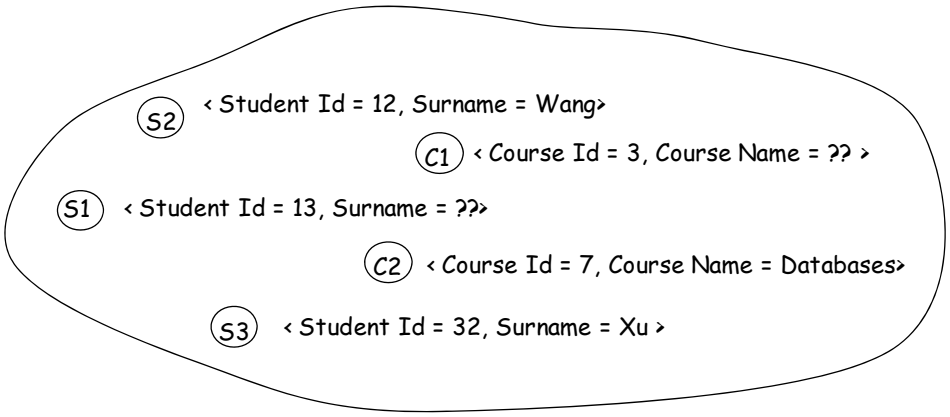
The instance of relation Student

We will use the term *relation* to denote the relation schema and the relation instance as a whole.

Definition - The instance of a database (also database instance) is the set of relation instances in the database. We will adopt the term *database* to denote the database instance and the database schema as a whole.

Incomplete information

A last concept we introduce in this lesson is the concept of *incomplete information*. Consider for instance the following ER instance, where instances Si represent students and Cj represent courses.



An ER instance

You see that for the student having Student Id = 13 we do not know the surname, and for the course with Course Id = 3 we do not know the Course name. This is a very frequent situation in our experience in common life, where sometimes we do not find all the information we need.

How do we represent this ER instance using two relations Student and Course in the relational model? We have to introduce the concept of null, meaning "I don't know this value". When for some attribute of some tuple we do not know the value, we write null in the corresponding cell. Notice that all the domains of attributes must be extended with a value *null* (we will see soon that this is not true for attributes composing the *primary key*). Therefore, the database instance is represented with the following relations.

Student

Student Id	Surname
12	Wang
13	null
32	Xu

Course

Course Id	Name
3	null
7	Databases

The ER instance represented with two null values

We conclude the lesson with an exercise.

Exercise 3.1 - Given the University database as represented below, extend the database schema with new requirements:

Student

Student Id	Surname
13	Batini
21	Xu
32	Smith

Exam

Student Id	Course Id	Grade
21	3	80
13	7	70

Course

Course Id	Name
3	Databases
7	Algorithms
4	Geometry

The University database

1. Add given names of students.
2. Add dates of exams, with day, month and year.
3. Add city of birth and country of birth of students.
4. Add professors, with professor id, surname, city of birth and country of birth.

avoiding redundancies and risks of inconsistencies.

Hint - Try to identify first if you have to create a new relation schema, or else if you can manage the change extending an existing relation schema; in this case identify first the relation schema.

Solution to Exercise 3.1

1. Add given names of students
Relation schema involved: *Student*

The new relation schema is
Student (StudentId, Given Name, Surname)

2. Add dates of exams, with day, month and year
Relation schema involved: *Exam*

The new relation schema is
Exam (StudentId, Course Id, Grade, Day, Month, Year)

3. Add city of birth and country of birth of students.
Relation schema involved: Student

In this third case the modification is not straightforward, and needs for some caution. In fact, if we extend the relation schema Student with two new attributes *City of Birth* and *Country of Birth*, we risk introducing redundancies in the relation instance. It is enough that several students are born in the same city, to give rise to several copies of the pair <city, country> in the relation.

A better solution is to extend Student with a unique attribute City of Birth, and add a new relation City, with two attributes City Name, and Country. The new pair of relation schemas is

Student (Student Id, Given Name, Surname, City of Birth)
City (Name, Country of Birth)

4. Add Professors, with Professor Id, Surname, City of Birth and Country of Birth
No previous relation involved

These requirements involve a completely new aspect of the reality of interest, that is professors. We can represent requirements with a new relation schema with four attributes, but we have to remember that a relation schema City (City Name, Country) has been created in the previous step, so we need only to define a single relation schema with three attributes:

Professor (Professor Id, Surname, City of Birth).

Part 3 – Lesson 3 – Introduction to integrity constraints

Look at the following database. Assume we are in a Chinese University. Grades are according to the Chinese rules, the range is [0 – 100]. Assume that CumLaude is a grade of excellence beyond grade 100, so it can be *yes* only when Grade = 100.

Student

StudentId	Surname
13	Batini
21	Xu
32	Smith
13	Wang

Exam

StudentId	CourseId	Grade	Cum Laude
21	3	80	no
13	7	70	yes
21	3	120	null
39	3	70	no
32	9	90	no

Course

CourseId	Name
3	Data Bases
7	Algorithms
4	Geometry

The usual database representing students, courses, and exams passed,
with a new attribute Cum Laude

Question 3.4 - Now, look carefully to the relation instances, initially one at a time, and then all together. Are you able to find "errors" in data, namely tuples or single values or pairs of values that do not represent a reasonable reality? Make a list of errors, written in natural language.

Discussion on Question 3.4

A possible list of errors is the following:

1. 120 in the third tuple of Exam is not a correct value of the domain of Grade (that is [0,100]).
2. In the 2nd tuple of Exam, the value yes of CumLaude is not compatible with value 70 for Grade.
3. 13 cannot be the StudentId of two students.
4. StudentId = 39 appears in Exam but does not appear as an Id of Student.
5. CourseId = 9 appears in Exam but does not appear as an Id of Course.

In a database, we need some mechanism that helps us to automatically discover what we have called *errors*. Instead of discovering ourselves errors, an highly time consuming activity, our interest is that the DBMS itself performs such check. So, we need to define *rules* that can be elaborated by the DBMS and that allow the DBMS to automatically check if the database is a correct representation of the reality: such rules are called in the relational model integrity constraints.

Definition - An *integrity constraint* is a logical property that has to be satisfied by all instances of a database. With the term *logical property*, we mean that its value can be *true* or *false*. True when the property is satisfied by all instances of the database, false otherwise.

In the relational model, we have four types of integrity constraints that are all present in the above list of errors. For each error, let us define the corresponding type of integrity constraint. In the next lessons, we will examine in depth the different types of constraints.

Integrity constraints in the exercise.

1. 120 in the third tuple of Exam is not a correct value of the Domain of Grade (that is [0,100]).
- This is called a *domain constraint*, as it expresses a condition on the membership of the value to a domain.

2. In the 2nd tuple of Exam, the value *yes* of CumLaude is not compatible with value 70 for Grade.
- This is called a *tuple constraint* as the condition is on the tuple.

3. 13 cannot be the StudentId of two students.
- This is called a *key constraint* and expresses a condition on the whole relation instance.

The above three types of constraints belong to the class of *intrarelation constraints* as they express logical conditions that must hold on single relations.

4. StudentId = 39 appears in Exam but does not appear as an Id of Student.
5. Course Id = 9 appears in Exam but does not appear as an Id of Course.

The above two constraints belong to the class of *interrelation constraints* as they express logical conditions referred to more than one relation instance, in our case referred to *two* relation instances. They are called *referential integrity constraints*.

Part 3 – Lesson 4 – Intrarelation constraints

Domain constraint

Definition - A domain constraint over an attribute A of a relation R expresses the property that values of A must belong to a given domain D(A).

Let us come back to our example, and assume that we are in a Chinese University, whose students are 20.000 and whose courses are 300.

Student

StudentId	Surname
13	Batini
21	Xu
32	Smith
13	Wang

Exam

StudentId	CourseId	Grade	Cum Laude
21	3	80	no
13	7	70	yes
21	3	120	null
39	3	70	no
32	9	90	no

Course

CourseId	Name
3	Data Bases
7	Algorithms
4	Geometry

The Chinese University database

The domain of the attribute StudentId is

Domain (Student Id) = [1….20.000], the set of integer values between 1 and 20.000.

The domain of the attribute Course Id is *Domain (Course Id) = [1…300].*

The domain of Grade is *Domain (Grade) = [1…100].*

The domain of CumLaude is *[true, false]* or *[yes, no].*

The domain of Surname is a bit more complex to define. I am not interested to examine in depth this point. We can assume that Surname can be any string of alphabetic characters, say, long no more than 20 characters.

Domain (Surname) = [any string of alphabetic characters long < = 20 characters]

Notice that for the definition of the domain of Surname we have adopted a description in natural language: again, I underline that I'm not interested in this course to analyze further in depth this topic. Notice also that all domains must be extended with the null character.

Tuple constraint

Definition - A tuple constraint over a relation R expresses a property that must be true for all tuples of R.

Coming back to our example

Student

StudentId	Surname
13	Batini
21	Xu
32	Smith
13	Wang

Exam

StudentId	CourseId	Grade	Cum Laude
21	3	80	no
13	7	70	yes
21	3	120	null
39	3	70	no
32	9	90	no

Course

CourseId	Name
3	Data Bases
7	Algorithms
4	Geometry

a tuple constraint is:

(In every tuple of Exam) when Grade is < 100 then Cum Laude cannot be yes.

Notice that we have used a restricted form of natural language for expressing the above tuple constraint. If you are interested in a more formal approach to the expression of tuple constraints, please refer to the Atzeni's book.

Key constraints

Key constraints are one of the most important, perhaps the most important concept in the relational model. They play in the relational model the same role of the identifier in the ER model. We first introduce keys by means of examples, and then we define them formally.

Consider the relation Student.

Student

StudentId	Surname
13	Batini
21	Xu
32	Wang
48	Wang

In this relation instance we see that given a StudentId, a unique Surname corresponds to it. Said in another way, two tuples cannot exist with the same StudentId and different surnames. Notice that the same property does not hold for surnames: we have two occurrences of surname Wang, which correspond to different Ids.

The property of StudentId to uniquely identify students is valid in *every* possible instance of Student. So we say that StudentId is a *key* of the relation Student. Similarly, Course Id is a key of Course, for the same reason.

Let us consider the relation Exam.

Exam

StudentId	CourseId	Grade	Cum Laude
21	3	80	no
13	7	70	no
21	7	100	yes
32	3	70	no
32	4	90	no

A relation with a more complex key

In this case, two different exams correspond to e.g. the student with Id = 32, and different exams correspond also to e.g. the course with Id = 7, so StudentId and CourseId alone cannot be a key for Exam. On the other hand, the pair <StudentId, CourseId> *is* a key of Exam, because in order to know the grade of an exam, we have to know both the student and the course the exam refers to.

We now move to a formal definition of key. To do so we have first to define the concept of *functional dependency*.

Consider again a relation Student.

Student

StudentId	Surname	City of Birth	Country of Birth
13	Batini	Rome	Italy
21	Xu	Harbin	China
32	Smith	Paris	France
48	Wang	Harbin	China

In this relation instance, and in all other relation instances of Student, as we have seen before in a similar example, a unique Surname, City of Birth, and Country of Birth correspond to each StudentId. We say that a functional dependency exists between StudentId and the set of attributes Surname, City of Birth, Country of Birth, and we write:

StudentId --> Surname, City of Birth, Country of Birth

Also the following functional dependency (among others) holds in the relation:

City of Birth --> Country of Birth

Definition - Given a relation schema R defined on attributes A1, A2,…, An, said B and C two disjoint subsets of the attributes, we say that a *functional dependency holds between B and C*, and write

B --> C

when *for each instance of R*, to each set of values <b1, b2, .., bk> of attributes in B, a unique set of values <c1, c2, …, ch> of attributes in C corresponds.

In the relation schema Student (StudentId, Surname, City of Birth, Country of Birth), meaning in all of its relation instances, the following functional dependencies hold

Student Id --> Surname, City of Birth, Country of Birth
City of Birth --> Country of Birth

Notice that in the above *specific* relation instance of the relation schema Student (StudentId, Surname, City of Birth, Country of Birth), other dependencies hold such as

Surname --> City of Birth
Surname --> Country of Birth

This is not true in general for all possible relation instances, as, e.g. two persons with the same surname can be born in different cities. E.g. Wang is a frequent surname in China, so in general we will have several Wangs born in different Chinese cities.

Definition - Given a relation schema R (A1, A2, …, An), when an attribute Ai or in general a group of attributes Ai1, ...,Aik of R is such that

1. Ai1, ...,Aik --> all other attributes in R

and

2. no subset of Ai1, ...,Aik has the same property

we say that *Ai1, ...,Aik* is a *key* of R.

The concept of key is *the most important concept in the relational model*. The value of a key allows to uniquely identify objects of the real world represented in R. For instance, given the relation Student above, we may want to retrieve *the unique* surname of *the unique* student with StudentId = 13, because we know that StudentId is a key, and so a unique surname may correspond to StudentId 13.

Student

StudentId	Surname	City of Birth	Country of Birth
13	Batini	Rome	Italy
21	Xu	Harbin	China
32	Smith	Paris	France
48	Wang	Harbin	China

Primary key

Assume that we are in China and assume also that a Social Security Number identifies all citizens in the country. Consider the new relation with the attribute Social Security Number added to it.

Student

StudentId	Social Security Number	Surname	City of Birth	Country of Birth
13	BTN57	Batini	Rome	Italy
21	X3472	Xu	Harbin	China
32	null	Smith	Paris	France
48	WNG54	Wang	Harbin	China

A relation with two keys

Now both for Student Id and for Social Security Number the following two functional dependencies hold

1. Student Id → Social Security Number, Surname, City of Birth, Country of Birth

2. Social Security Number → Student Id, Surname, City of Birth, Country of Birth

Therefore, both StudentId and Social Security Number are keys of the relation. However, contrary to what happens for the Social Security Number, only StudentId is always specified, and so only StudentId allows to uniquely identifying students.

Definition - We call *primary key* a key whose values are always specified, namely, are different from null.

We want to emphasize a point that we addressed before in the lesson. Look at the following relation.

Student

StudentId	Social Security Number	Surname	City of Birth	Country of Birth
13	BFG3RTK	Batini	Rome	Italy
21	null	Xu	Harbin	China
32	GRT5YHF	Smith	Paris	France
48	SDE5IKL	Wang	Harbin	China

A specific relation instance where more functional dependencies are valid

In the specific relation instance shown above also the following functional dependency holds

Surname --> Student Id, Social Security Number, City of Birth, Country of Birth

but this is not true in general, for all possible instances!!!

Before concluding our discussion on keys, we aim to introduce a graphical notation for them. Consider the database

Student

Student Id	Social Security Number	Surname	City of Birth	Country of Birth
13	BFG3RTK	Batini	Rome	Italy
21	null	Xu	Harbin	China
32	GRT5YHF	Smith	Paris	France
48	SDE5IKL	Wang	Harbin	China

Exam

Student Id	Course Id	Grade
13	3	80
13	7	70
21	7	60

A database with two relations

We know we may represent the two relation schemas as

Student (Student Id, Social Security Number, Surname, City of Birth, Country of Birth)
Exam (Student Id, Course Id, Grade)

We may represent the primary keys of the two relation schemas as follows

Student (Student Id, Social Security Number, Surname, City of Birth, Country of birth)
Exam (Student Id, Course Id, Grade)

underlying all the attributes in a relation schema that take part to the key.

Question 3.5 - Find keys of relation schemas in the following database schema

Student (Student Id, Given Name, Surname, City of Birth)
City (Name, Country)
Country (Name, Continent)
Course (Course Id, Name, Year of Enrollment)
Professor (Professor Id, Given Name, Surname)
Teaches (Professor Id, Course Id, Number of Hours)

assuming that more than one professor can teach a course.

Answer to Question 3.5

Student (Student Id, Given Name, Surname, City of Birth) City (Name, Country) Country (Name, Continent) Course (Course Id, Name, Year of Enrollment) Professor (Professor Id, Given Name, Surname) Teaches (Professor Id, Course Id, Number of hours)

Question 3.6 - Assume now we represent in the following schema all Chinese university students

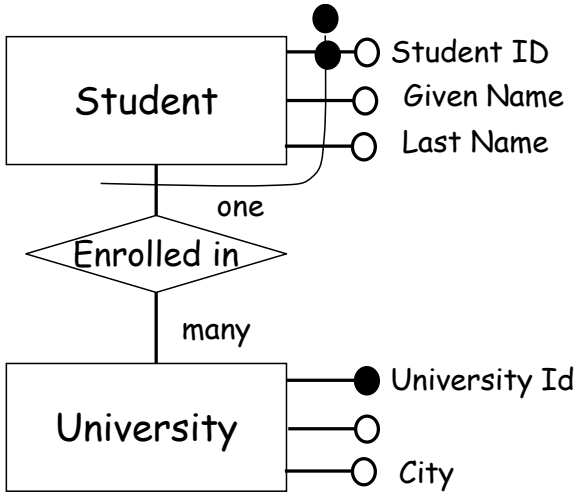
Student (Student Id, University Id, Given Name, Last Name) University (University Id, Name, City)

Find the two primary keys.

Answer to Question 3.6

Student (<u>Student Id, University Id</u>, Given Name, Last Name) University (<u>University Id,</u> Name, City)

Notice that the key is similar in structure to the external identifier of Student for the ER schema in the figure.



The Student – University schema represented in the ER model

Part 3 – Lesson 5 – Interrelation constraints: referential integrity constraints

Referential integrity constraints

Student

StudentId	Surname
13	Batini
21	Xu
32	Smith
13	Wang

Exam

StudentId	CourseId	Grade	Cum Laude
21	3	80	no
13	7	70	yes
21	3	120	null
39	3	70	no
32	9	90	no

Course

CourseId	Name
3	Data Bases
7	Algorithms
4	Geometry

A database with three relations

Consider the above database and the two errors we have identified, and we did not discuss so far,

1. StudentId = 39 appears in Exam but does not appear as an Id of Student.
2. CourseId = 9 appears in Exam but does not appear as an Id of Course.

Why we do not accept that the student with Student Id = 39 appears in Exam and does not appear in Student? The reason is that relations such as Student play a role of *registry*: all Student Ids appearing somewhere in the database instance, must appear in Student. So, given, as an example, the schema

Student (Student Id, Surname)
Exam (Student Id, Course Id, Grade, CumLaude)
Course (Course Id, Name)

we say that the following *referential integrity constraint* holds

Exam (StudentId) --> Student (StudentId)

meaning that if a value v of StudentId exists in Exam, it must exist also in Student. Notice that StudentId is the key of Student. This is the reason why a referential integrity constraint is also called *external key*, meaning that the value *v* externally identifies tuples of Student.

Notice the following roles in the referential integrity constraint

Exam (StudentId) --> Student (StudentId)

Primary relation or Registry relation Secondary Relation

Question 3.7 - Find all referential integrity constraints defined in the following database.

Student

StudentId	Surname
13	Batini
21	Xu
32	Smith
13	Wang

Exam

StudentId	CourseId	Grade	Cum Laude
21	3	80	no
13	7	70	yes
21	3	120	null
39	3	70	no
32	9	90	no

Course

CourseId	Name
3	Data Bases
7	Algorithms
4	Geometry

Database with two registry relations, Student and Course

Answer to Question 3.7

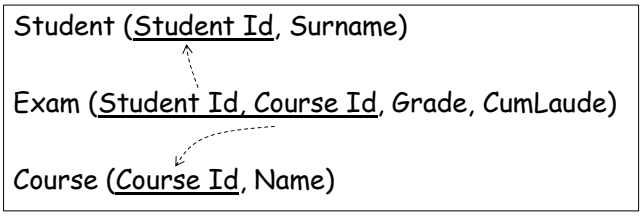
Exam (Student Id) → Student (Student Id)
Exam (Course Id) → Course (Course Id)

Graphical notation for referential integrity constraints

For referential integrity constaints, besides the notation

Exam (StudentId) --> Student (StudentId)
Exam (CourseId) --> Course (CourseId)

we can also use a graphical notation



Graphical notation for integrity constraints

A more complex example of referential integrity constraint

Assume that cars in a Country are identified in their car plates with

a. a RegionCode, and
b. a progressive number in the set of cars of the Region.

So, in Italy a plate such as

Tuscany - 65456

identifies a car that is registered in the Tuscany Region and has number 65456. Cars can be represented with a relation schema defined as

Car (Region Code, Number, Make, Color, Number of Seats)

Assume now that we want to represent in another relation the accidents between pairs of cars, and that a pair of cars can be involved in only one accident.

Such relation can be

Accident (Region Code1, Number1, Region Code2, Number2, Place of Accident, Date)

as we know that each pair of cars can be involved in only one accident.

Now, given the two relation schemas

Car (Region Code, Number of Car, Make, Color, Number of Seats) Accident (Region Code1, Number1, Region Code2, Number2, Place of Accident, Date)

Question 3.8 - Find the keys of the two relations

Question 3.9 – Find referential integrity constraints

Answer to Question 3.8

Car (Region Code, Number, Make, Color, Number of Seats)

Accident (Region Code1, Number1, Region Code2, Number2, Place of Accident, Date)

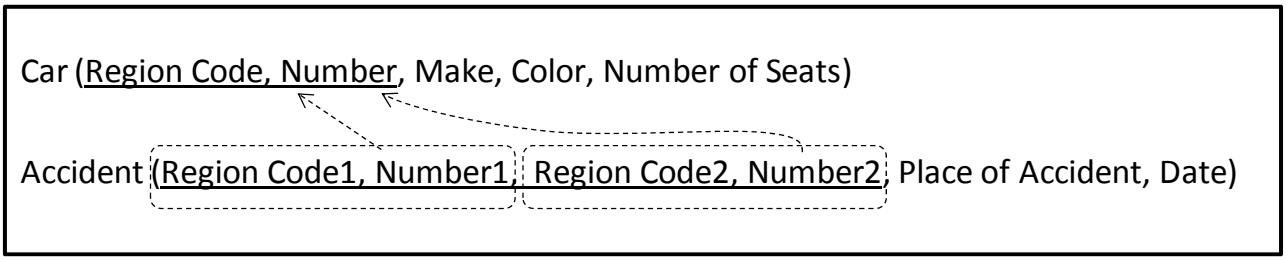
Notice that

Region Code1, Number1, Region Code2, Number2

Is the key of the relation schema Accident, as we know the two cars can have in only one accident.

Answer to Question 3.9

We have two symmetric referential integrity constraints between the two plates of the two cars involved in the accident and the plate of the car in the Car relation.



Two referential integrity constraints

Exercise 3.2 - Given the following database schema

Student (StudentId, Given Name, Surname, Date of Birth, City of Birth, Country of Birth)

Country (Name, Continent)

Course (CourseId, Course Name, Semester, Year of Enrollment)

Professor (ProfId, SocialSecurityNumber, Given Name, Surname, Date of Birth, City of Birth, Country of Birth)

Teaches (CourseId, ProfId)

Enrolled (StudentId, CourseId)

Passed (StudentId, CourseId, Date, Grade)

CourseSchedule (CourseId, Day of Week, Room, Building, Start Hour, End Hour)

Room (Room Number, Building, Floor, Number of Seats)

assume that

1. A professor can teach different courses, and a course can be taught by only one professor.
2. Courses can be taught in different days of week, in different rooms of different buildings.
3. Two rooms in different buildings may have the same number, two rooms in the same building have different numbers.

Find and represent keys and referential integrity constraints.

Solution to Exercise 3.2 (keys)

Student (StudentId, Given Name, Surname, Date of Birth, City of Birth, Country of Birth)

Country (Name, Continent)

Course (CourseId, Course Name, Semester, Year of Enrollment)

Professor (ProfId, SocialSecurityNumber, Given Name, Surname, Date of Birth, City of Birth, Country of Birth)

Teaches (CourseId, ProfId)

Enrolled (StudentId, CourseId)

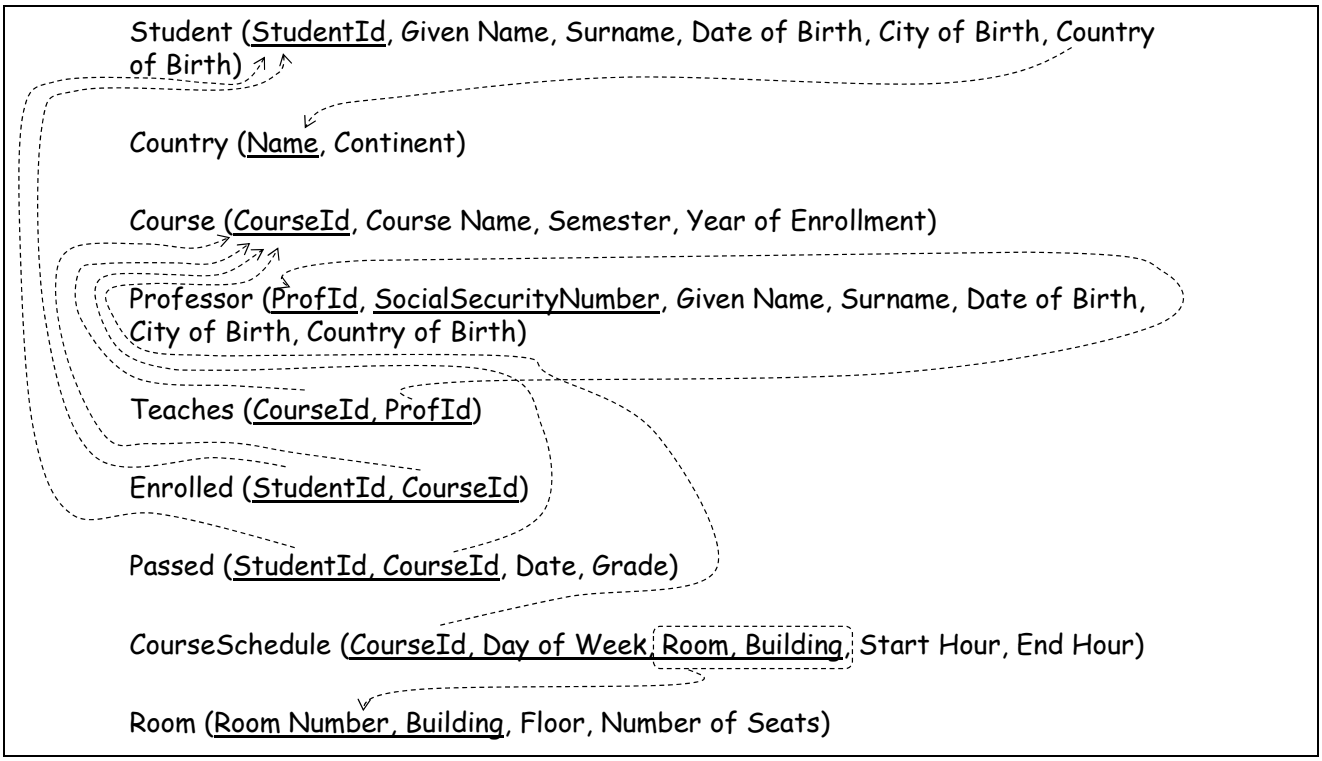
Passed (StudentId, CourseId, Date, Grade)

CourseSchedule (CourseId, Day of Week, Room, Building, Start Hour, End Hour)

Room (Room Number, Building, Floor, Number of Seats)

Furthermore ProfId is the primary key of Professor.

Solution to Exercise 3.2 - referential integrity constraints



Part 3 – Lesson 6 – Normalization

Consider the following relation, that represents Chinese and Foreign students of a Chinese University.

Student

StudentId	Surname	City of Birth	Country of Birth	Continent of Birth
37	Batini	Rome	Italy	Europe
41	Rossi	Rome	Italy	Europe
53	Wang	Harbin	China	Asia
26	Xu	Harbin	China	Asia
14	Smith	Rome	Italy	Europe
39	Maigret	Paris	France	Europe
76	Hollande	Paris	France	Europe

Motivating example for normalization

We know, and we see, that in this schema we have redundancies, e.g. the pair <Paris, France> appears two times.

Question 3.10 - Find the functional dependencies in the schema.

Answer to Question 3.10

Functional dependencies are

1. Student Id → Surname, City of Birth, Country of Birth, Continent of Birth
2. City of Birth → Country of Birth, Continent of Birth
3. Country of Birth → Continent of Birth

Question 3.11 - Do you have any idea on transforming the schema into a set of relational schemas in such a way that we succeed in removing redundancies?

Hint: start with looking for a transformation that removes the redundancies on <Harbin, China> and <Paris, France>.

Answer to Question 3.10

As a solution of the exercise, we can generate, among others, the following schemas

Schema 1
Student (Student Id, Surname, City of Birth)
City (Name, Country, Continent)

Schema 2
Student (Student Id, Surname, City of Birth)
City (Name, Country)
Country (Name, Continent)

Let us generate the functional dependencies in the two schemas and compare their structure.

Schema 1
Student (Student Id, Surname, City of Birth)
1. Student Id → Surname, City of Birth

City (Name, Country, Continent)
1. Name → Country, Continent
2. Country → Continent

Notice that in Schema 1 potential redundancies are present in the City relation.

Schema 2
Student (Student Id, Surname, City of Birth)
1. Student Id → Surname, City of Birth

City (Name, Country)
1. Name → Country

Country (Name, Continent)
1. Name → Continent

Notice that in Schema 2 no potential redundancy exists.

Definition - A *key dependency* is a dependency whose left hand part is a key.

All relations in Schema 2 *have only key dependencies*. We say that Schema 2 is *in normal form* (or *normalized*), while Schema 1 is un-normalized.

Definition - A relation schema R is in normal form, or better, is in *Boyce Codd normal form* (BCNF) if all functional dependencies in R are key dependencies.

Definition - A database schema is *in Boyce Codd normal form (BCNF)* if all its relation schemas are in BCNF.

The database instance corresponding to Schema 2 is

Student

StudentId	Surname	City of Birth
37	Bqtini	Rome
41	Rossi	Rome
53	Wang	Harbin
26	Xu	Harbin
14	Smith	Rome
39	Maigret	Paris
76	Hollande	Paris

City

Name	Country of Birth
Rome	Italy
Harbin	China
Paris	France

Country

Country of Birth	Continent of Birth
Italy	Europe
China	Asia
France	Europe

A database in Boyce Codd Normal Form

Looking at the above database instance, we notice that we have eliminated all redundancies. And the reason is that, as the definition of normal form says, all dependencies in relations are key dependencies, so *every pair of values can appear only once in a relation instance*.

Exercise 3.3 – Given the following five schemas, check which schemas are in BCNF.

Schema 1
Student (Student Id, Surname, Course Id, Course Name, Grade of Exam, Date of Exam)

Schema 2
Student (Student Id, Surname, Course Id, Course Name)
Exam (Student Id, Course Id, Grade of Exam, Date of Exam)

Schema 3
Student (Student Id, Surname)
Exam (Student Id, Course Id, Course Name, Grade of Exam, Date of Exam)

Schema 4
Student (Student Id, Surname)
Exam (Student Id, Course Id, Grade of Exam, Date of Exam)
Course (Course Id, Course Name)

Schema 5
Student (Student Id, Surname)
Exam1 (Student Id, Course Id, Grade of Exam)
Exam2 (Student Id, Course Id, Date of Exam)
Course (Course Id, Course Name)

Discussion on Exercise 3.3

Let us identify the functional dependencies in the five cases.

Schema 1

Student (Student Id, Surname, Course Id, Course Name, Grade of Exam, Date of Exam)
Student Id, → Surname
Course Id → Course Name
Student Id, Course Id → Grade of Exam, Date of Exam

The schema is not in BCNF, since relation Student violates BCNF

Schema 2

Student (Student Id, Surname, Course Id, Course Name)
Student Id → Surname
Course Id → Course Name

Exam (Student Id, Course Id, Grade of Exam, Date of Exam)
Student Id, Course Id → Grade of Exam, Date of Exam

The schema is not in BCNF, since relation Student violates BCNF

Schema 3

Student (Student Id, Surname)
Student Id → Surname

Exam (Student Id, Course Id, Course Name, Grade of Exam, Date of Exam)
Student Id, Course Id → Grade of Exam, Date of Exam
Course Id → Course Name

The schema is not in BCNF, relation Exam violates BCNF

Schema 4

Student (Student Id, Surname)
Student Id → Surname

Exam (Student Id, Course Id, Grade of Exam, Date of Exam)
Student Id, Course Id → Grade of Exam, Date of Exam

Course (Course Id, Course Name)
Course Id → Course Name

The schema is in BCNF, all dependencies in relation schemas are key dependencies.

Schema 5

Student (Student Id, Surname)
Student Id → Surname

Exam1 (Student Id, Course Id, Grade of Exam)
Student Id, Course Id → Grade of Exam

Exam2 (Student Id, Course Id, Date of Exam)
Student Id, Course Id → Date of Exam

Course (Course Id, Course Name)
Course Id → Course Name

The schema is in BCNF, all dependencies in relation schemas are key dependencies. In this case we have used two relation schemas to represent exams, and this is a bit counterintuitive. We will see in Part 5 – Logical Design how to deal with these issues.

How to Normalize an Un-normalized Schema

We have seen that normalization is a "good" property of a schema. Sometimes it may happen that checking for BCNF, some schema reveals un-normalized. How can we transform the schema in a new schema, with the same information content, but normalized in BCNF? A procedure for database schema normalization is the following.

1. Identify relation schemas that are not normalized.

2.For each un-normalized relation schema R:

Decompose the un-normalized relation schema R, separating in different relations schemas functional dependencies that violate BCNF.

Until all the resulting relation schemas are in BCNF.

Let us apply the normalization procedure to the following database schema and instance, made of a unique relation.

Student

Student Id	Surname	City of Birth	Country of Birth	Continent of Birth
37	Batini	Rome	Italy	Europe
41	Rossi	Milano	Italy	Europe
53	Wang	Harbin	China	Asia
26	Xu	Pecking	China	Asia
14	Smith	Rome	Italy	Europe
39	Maigret	Paris	France	Europe

An un-normalized relation

Functional dependencies are
Student --> Surname, City of Birth, Country of Birth, Continent of Birth
City of Birth --> Country of Birth, Continent of Birth
Country of Birth --> Continent of Birth

Input to the normalization procedure

Relation Student (Student Id, Surname, City of Birth, Country of Birth, Continent of Birth)
with dependencies
Student --> Surname, City of Birth, Country of Birth, Continent of Birth
City of Birth --> Country of Birth, Continent of Birth
Country of Birth --> Continent of Birth

First normalization step: separate Country of Birth --> Continent of Birth

Output of the normalization step: two relation schemas

1. Relation Student (Student Id, Surname, City of Birth, Country of Birth)

with dependencies
Student --> Surname, City of Birth, Country of Birth
City of Birth --> Country of Birth

2. Relation Country (Country of Birth, Continent of Birth)

with dependency
Country of Birth --> Continent of Birth

Second normalization step: separate City of Birth --> Country of Birth

Output of the normalization step – three relation schemas

1. Relation Student (Student Id, Surname, City of Birth)

with dependency
Student Id--> Surname, City of Birth

2. Relation City (City of Birth, Country of Birth)

with dependency
City of Birth --> Country of Birth

3. Relation Country (Country of Birth, continent of Birth)

with dependency
Country of Birth --> Continent of Birth

We have achieved BCNF for all relation schemas!!! The database schema is in BCNF

The final schema is

Student (Student Id, Surname, City of Birth)
City (City of Birth, Country of Birth)
Country (Country of Birth, Continent of Birth)

that for clarity we can change into a new schema with some attribute name modified.

Student (Student Id, Surname, City of Birth)
City (Name, Country of Birth)
Country (Name, Continent of Birth)

Transformations without loss of information content

Everything is OK! No,…….be careful when you transform a relation schema R(A,B,C) into two relation schemas R1 and R2 with the goal of obtaining a normalized schema.

As we have seen in the first lesson on the relational model, some transformations of a relation R that decompose R into two (or more) relations are *reversible*, namely they can be performed in the reverse direction allowing to reconstruct the original relation, other decompositions result in a

loss of information, and the original relation cannot be reconstructed. We provide now a sufficient condition for characterizing the transformations that are without loss of information.

Property of decomposition without loss of information content - Given a relation schema R, if we decompose a relation instance of R into two two instances of relation schemas R1 and R2, the original instance of R can be reconstructed without loss of information content if $Attributes\ (R1) \cap Attributes\ (R2)\ is\ a\ key\ of\ R1\ or\ a\ key\ of\ R2$

In previous normalization steps we always selected functional dependencies A → B (where A was a key) that produced a new relation R(A,B), so we always respected the property of decomposition without loss of information. E.g., when we have transformed

Student (Student Id, Surname, City of Birth, Country of Birth, Continent of Birth)

using the dependency *Country of Birth --> Continent of Birth* into

Student (Student Id, Surname, City of Birth, Country of Birth)

Country (Country of Birth, Continent of Birth)

Attributes (Student) $\cap$ $\mathrm{Attributes\ (Country)}$ is equal to *Country of Birth*, the key of Country, and so the corresponding decomposition is without loss of information.

Concepts defined in Part 3

Part 3 - Relational Model

- Relation
  - Schema
  - Instance
  - Tuple
- Attribute
  - Domain of an Attribute
  - Value of an Attribute
- Data Base
  - Schema
  - Instance
- Incomplete information
- Integrity Constraint
  - Intra Relation Int. Constraint
    - Domain Constraint
    - Tuple Constraint
    - Key
    - Primary Key
  - Inter Relation Int. Constraint
    - Referential Integrity
- Rel. Data Base Quality: Normalization
  - Boyce Codd Normal Form
  - Normalization Process

Part 3 – Exercise assignment

Solve exercises from 2.1 to 2.8 of Chapter 2 of Atzeni's book. Then compare your solutions with solutions provided in the course site.