Relational Logic

1. Introduction

The relational model was proposed by E.F. Codd in the early seventies. The model was meant to describe the structure of the data and the operations on the data. The model became popular because efficient software implementations have been realized partly by the hardware which became faster and faster and more and more powerful. But an important reason also lies in its simplicity: the data structure is the one of the table, a well-known and often used representation of data. Apart from that, the model has a strong theoretical basis, namely the mathematical relation.

We will look at this theoretical foundation more closely here. In the relational theory a table is seen as a (mathematical) relation, thus a set of rows. We will specify this later on. This also explains the name of the model.
There are two ways of describing the relational model in terms of mathematics. The first way is called Relational Algebra. In the relational algebra operations on relations (tables) are defined, that is to say, rules on how to form new relations from existing ones. The second way closely follows the preceding theory because this is based on the predicate logic, also called predicate calculus. That is why in literature this is called either Relational Calculus or Relational Logic.

We will discuss this second method here.

In principle a data model contains the following parts:

- A data structure
- A language to define the data structure
- A language to manipulate the data
- Limitation rules on safety and integrity

We will limit ourselves to a description of the data structure and a mathematical description of the language to manipulate data, especially the part to retrieve data, to view data.

2 Relational Structure

A table in a relational database is in fact a relation. Such a relation is however defined in a special way, namely by

- The name of the relation
- A collection of column names (attribute names)
- A domain for each column

The definition is actually a type-declaration for a possible concrete table or relation.
Such a relation is then a set of rows (tuples), consisting of attributes (values) of an object from reality.
In principle a domain is a collection of values from which the attributes can be chosen, usually a data type.
The relation is now determined by the definition, also called header and the collection of tuples called the body.

Let us write this down in a mathematical notation.

We note the header of a relation as $R(A_1, A_2, ..., A_n)$, in which R is the name of the relation and A_i the name of the i-th column of the relation.
With every column A_i there is a domain D_i, a collection from which the attributes (column values) can be chosen.

A concrete relation, the body, is then a subset of the Cartesian product
$D_1 \times D_2 \times \ldots \times D_n$ otherwise $R = \{(a_1, a_2, ..., a_n) | a_i \in D_i\}$.

Example 1:

Let us look at the relation *Student(studno,* studname, study, address, residence). This way we have determined the name of the relation and names of the columns. Besides we know that the relation consists of 5-tuples, rows of five attributes. We will not define the domain explicitly here. We will give another example of a concrete relation Student:

Student = {(278123, Susan Groen, Computer Science, Decemberstraat 12, Almere), (314182, Paul Haenen, Computer Science, Parkstraat 78, Haarlem), (392044, Tim Bruins, Mathematics, Rustenburgerstraat 23I , Amsterdam) }

Of course, it is much clearer to present this in the form of a table. We will do so from now on.

With the definition of a relation the primary key is often mentioned as well. For the database system this is of essential importance. This happens by underlining the header, the primary key in the definition: *Student*(*studno*, studname, study, address, residence).

This integrity rule cannot be put down mathematically.
You can describe the referential integrity mathematically. This rule requires all values in a strange key column to appear in the accompanying primary key column.

In other words: the collection of strange key values is a subset of the set of primary key values. This integrity rule is therefore also called subset rule.

3. Relational operators

The theoretical model of Codd contains a number of actions, operators with which relations can be edited or formed into new relations.

We use the terms unary and binary operators. The binary operators are based on the set operators which we already know: the union, the intersection, the difference and the product. The fact that we can use these operators is of course logical, for relations are sets.

We must, however, put some restrictions on the operators to lead to useful results in a relational database.

There are also a number of special relational operators: projection, restriction, the join and the division. We will discuss these in this section later.

We must put restrictions on the set of operators because we can only edit relations that contain the same kinds of tuples. Firstly the number of columns must be equal. But the headers, the definitions should correspond too. There is no point in adding tuples with a number in the first place to tuples with a name in the first place. ‘Set theoretically’ there is no objection but it give a confusing result in a relational database. The relations must meet the following requirements:

1. They must have the same set of column names in the definitions and in the same order
2. The columns with the same names are defined on the same domain

We call such relations **union-compatible.**

The **union, intersection** and the **difference** of two union-compatible relations are relatively simple. In the language of predicate logic and set theory we write the following:

Given two union-compatible relations R and S, then the union of R and S is:

$$R \cup S = \{t | t \in R \vee t \in S\}$$

the intersection of R and S is:

$$R \cap S = \{ t \mid t \in R \wedge t \in S \}$$

the difference of R and S is:

$$R - S = \{ t \mid t \in R \wedge t \notin S \}$$

Example 2:

Employee x Department

eno	ename	salar	dno	dno	dname	location
236	Smit	1200	10	10	Sales	Andijk
236	Smit	1200	10	20	Production	Bovenkarspel
236	Smit	1200	10	30	Management	Hoorn
243	Tuip	1450	20	10	Sales	Andijk
243	Tuip	1450	20	20	Production	Bovenkarspel
243	Tuip	1450	20	30	Management	Hoorn
257	Koster	1680	10	10	Sales	Andijk
257	Koster	1680	10	20	Production	Bovenkarspel
257	Koster	1680	10	30	Management	Hoorn
283	Baas	3270	30	10	Sales	Andijk
283	Baas	3270	30	20	Production	Bovenkarspel
283	Baas	3270	30	30	Management	Hoorn
301	Arts	1450	20	10	Sales	Andijk
301	Arts	1450	20	20	Production	Bovenkarspel
301	Arts	1450	20	30	Management	Hoorn

The natural join is then the set

$\{(E.eno, E.ename, E.salary, E.dno, D.dname, D.location) \mid$
$(E.eno, Rename, E.salary, E.dno) \in W \wedge (D.dno, D.dname, D.location) \in D \wedge$
$E.dno = D.dno\}$

In table form this yields the following result:

eno	**ename**	**salary**	**dno**	**Dname**	**location**
236	Smit	1200	10	Sales	Andijk
243	Tuip	1450	20	Production	Bovenkarspel
257	Koster	1680	10	Sales	Andijk
283	Baas	3270	30	Management	Hoorn
301	Arts	1450	20	Production	Bovenkarspel

4. Additional examples

We will further demonstrate the use of relational logic with a number of example queries (requests for information). In these queries we will formulate the mathematical expression which will describe the outcome of the query. There will be combinations of restrictions and projections in these queries. The join restriction will be discussed as well.

In the examples we will use a database with the relations *Person(pno,* pname), *Committee*(*cno,* cname, chair) en *Member_of* (cno, pno).
The column chair of the relation Committee contains the

Personal number (pno) van de person who is now chair. The relation Member_of consists of a list of the committees with the persons that are member.

The contents of the relations is shown in the following table:

Person	
pno	**pname**
1023	Brown
1111	Smith
1373	Cook
1447	Green
1529	Zoerb
1919	Yang

Committee		
cno	**cname**	**chair**
101	Appeals	1023
202	Planning	1919
303	Budget	1373
404	Awards	1373
505	Research	1447

Member_of	
cno	**pno**
101	1023
202	1919
303	1529
404	1373
505	1447
202	1023
303	1023
404	1023
505	1023
505	1529
404	1447
101	1529
202	1373
202	1447
303	1111
303	1373

Example 3:
We will first show an example of a combination of a projection and a restriction on one table. We need the implicit notation of the set with the personal numbers of the members of committee 202.

For this we can choose in the relation Member_of those tuples of which the cno = 202 (a restriction) and then we take the projection on the pno.
So we are looking for the set

$$\{t.pno \mid t \in Member\_of \wedge t.cno = 202)\}$$

Example 4:

Next we look at a situation in which the restriction is determined by the occurrence of tuples in another table. You might call this a join restriction. However we will not make the complete join, that is we do not join the tables into one big new table. We want to choose tuples from one table and look at the other table for the restriction.

Because the restriction is determined by presence (or absence) of another tuple in the other table we need the existential quantifier. We will give an example.

We need the implicit notation of the set with all data of the members of committee 202. We look for data of members of committee 202 so we look for tuples of the relation (table) Person.

But how do we know if a person is a member of committee 202? We will find out by looking at the relation Member_of: if there is a tuple in the relation Member_of with the same pno as persontuple and cno = 202.

The condition for this restriction is therefore a proposition with an existential quantifier and the join condition t.pno = m.pno is necessary to join the right tuples of the two relations:

$$\{m \in Person \mid (\exists t : t \in Lid\_Van : t.pno = m.pno \wedge t.cno = 202)\}$$

The explicit notation will be in this case:

{(1919, Yang), (1023, Brown), (1373, Cook), (1447, Green)}

Example 5:

Finally we will look at an example in which we need all three relations to find the right restriction. We still need the join condition between the two mutual relations.

We want to determine the contents of the following set:

$$\{m \in Person \mid (\exists\ t, u: t \in Member\_of \wedge u \in Committee: t.pno = m.pno \wedge t.cno = u.cno \wedge u.cname = \text{'Research'}\}$$

We must find those persons of which the pno appears in the Member_of.
With that pno in Member_of there goes a cno for which there must be an appropriate row in Committee. But this row must contain the cname Research then.
In reverse order we are looking for persons who are a member of the Research Committee.
And they are:

pno	**pname**
1023	Brown
1447	Green
1529	Zoerb