

Databases from a logical perspective

18 oktober 2016

In this exercise session we start from the concept of relational databases (in the rest of this assignment: database) and we investigate the aspects of databases from a logical perspective:

- Different logical visions for databases
- Modelling information in databases in logic
- The representation of queries on a database in logic as a modelling exercise.

1 A Database as a logical structure

Consider the next database (cf the course): From a logical perspective we

Instructor		Enrolled		Grade			PassGrade
Ray	CS230			Sam	CS148	AAA	
Hec	CS230	Jill	CS230	Bill	CS148	D	AAA
Sue	M100	Jack	CS230	Jill	CS148	A	AA
Sue	M200	Sam	CS230	Jack	CS148	C	A
Pat	CS238	Bill	CS230	Flo	CS230	AA	B
Prerequ ^a		May	CS238	May	CS230	AA	C
CS230	CS238	Ann	CS238	Bill	CS230	F	
CS148	CS230	Tom	M100	Ann	CS230	C	
M100	M200	Ann	M100	Jill	M100	B	
		Jill	M200	Sam	M100	AA	
		Sam	M200	Flo	M100	D	
		Flo	M200	Flo	M100	B	

^aPrerequ(a,b)
means a is a prerequisite for b

Figuur 1: The student database

can directly see this database as a structure I over a vocabulary Σ .

$$\Sigma = \{Instructor/2, Grade/3, Enrolled/2, Prerequ/2, PassingGrade/1, \\ Ray, Hec, \dots, CS230, \dots, AAA, \dots, Jill, \dots\}$$

The domain is

$$Domein_I = \{Ray, Hec, \dots, CS230, \dots, AAA, \dots, Jill, \dots\}$$

The interpretation of I consists of

- The interpretation of all constants: $Ray^I = Ray, \dots, CS230^I = CS230, \dots, AAA^I = AAA, \dots$
- The relation of tuples of domain elements $Instructor^I, Grade^I, \dots$ as in Figure 1

I is the Herbrand structure.

Exercise 1. Formulate the next queries in Σ en evaluate them in structure I . All queries are *simple*: the answer is *true* or *false*. On Toledo you can find an example file in which you can fill the queries and use the IDP system to answer them.

1. *Is there a prerequisite for CS238?*
2. *Has May passed for CS230 ?*
3. *Are all the courses which are (direct) prerequisite for M100 instructed by Ray?*
4. *Has everyone that is enrolled in CS230 passed for at least one course instructed by Sue?*
5. *Did John do an exam for CS230?*
6. *Are there students taking a course from every instructor?*
7. *Is there are student who is enrolled in exactly 1 course?*

Exercise 2 An n -ary query is a symbolic definition of a relation of the form $\{(x_1, \dots, x_n) : A\}$. On Toledo you can find an example file in which you can write the queries and use the IDP system to answer them. Construct the following n -ary queries:

1. *All tuples (x, y) such that student x has passed for the course y .*
2. *All students who follow exactly one course*
3. *All students who are enrolled in CS230 and have passed for all the courses they did up til now.*

Exercise 3. An essential part of the database are the integrity constraints. The database is subject to these constraints. These need to be checked when the database is changed and need to prevent the creation of invalid relations. Express the following integrity constraints in Σ (on Toledo you can find an example file in which you can fill in the constraints and call the IDP system to test if they are satisfied in the given database).

1. *No subject is a (direct) prerequisite for itself*
2. *Nobody can have more than one grade for a course.*
3. No student can be an instructor. (A student is someone who is enrolled in a course)
4. *Everyone who is enrolled for a course needs to have passed all courses which are (direct) needed prerequisites for that course.*

2 SQL

The query language SQL is a widespread language for formulating queries on database systems. SQL is based on the principles of predicate logic but has a strongly dissimilar syntax. Nevertheless it isn't difficult to see the connection to logic.

In an SQL database the vocabulary (called the *database-scheme*) is given in the form of a number of table-declaration of the form:

$$table(field_1, field_2, \dots, field_n)$$

Here *table* is the name of the table and *field_i* is the name of the *i*th column of the table. E.g. *Grade(student, vak, score)*, *Enrolled(student, score)*.

Exercise 5. Given this vocabulary we can translate SQL-queries to queries in logic. Translate the next SQL query to logic (you can do this on paper instead of in IDP)

```
SELECT student
FROM Grade AS Gr1
WHERE NOT EXISTS ( SELECT score
FROM Grade AS Gr2
WHERE Gr1.student = Gr2.student AND NOT (score = 'AA' OR score = 'AAA'))
```

Different database schemes can correspond to the same domain model, and even the same database scheme can be translated to different logical vocabularies. The chosen scheme has effect on the simplicity of the SQL queries, and in the same way the logical vocabulary has effect on the simplicity of modelling or execution speed of the inferences. We propose two possible ways to convert a scheme into a vocabulary.

- The “classical” way: we choose an *n*-ary predicate symbol per table with *n* columns.
- The “object oriented” way: we choose the table name *table* as a unary type-predicate and the column names as *field_i* as binary relations. For each row of the table a new “identifier” is created who represent the objects.

The SQL-table *Grade(student, subject, grade)* can be represented as a ternary relation as in the student database. The alternative is to view this table as a set of relations in Figure 2 for the predicate symbols *Grade/1*, *G2Student/2*, *G2Vak/2* and *G2Score/2*.

Exercise 5 (continued). Translate the previous SQL query to this new vocabulary.

Exercise 6. Take a table for representing employees in a company: *Employee(id, surname, name, street, houseNb, city, zip, birthDate, job, salary, nbOfServedYears)*. This is a relation with 11 arguments. Suppose we want to know which employees earn more than 100.000 a year. In SQL this can be written down as:

```
SELECT id
FROM Werknemer
WHERE Werknemer.loon ≥ 100.000;
```

Grade			⇒	Grade ^I		G2Student ^I		G2Vak ^I		G2Score ^I	
Sam	CS148	AAA		G1	G1	Sam	G1	CS148	G1	AAA	
Bill	CS148	D		G2	G2	Bill	G2	CS148	G2	D	
Jill	CS148	A		G3	G3	Jill	G3	CS148	G3	A	
Jack	CS148	C		G4	G4	Jack	G4	CS148	G4	C	
Flo	CS230	AA		G5	G5	Flo	G5	CS230	G5	AA	
May	CS230	AA		G6	G6	May	G6	CS230	G6	AA	
Bill	CS230	F		G7	G7	Bill	G7	CS230	G7	F	
Ann	CS230	C		G8	G8	Ann	G8	CS230	G8	C	
Jill	M100	B		G9	G9	Jill	G9	M100	G9	B	
Sam	M100	AA		G10	G10	Sam	G10	M100	G10	AA	
Flo	M100	D		G11	G11	Flo	G11	M100	G11	D	
Flo	M100	B	G12	G12	Flo	G12	M100	G12	B		

Figure 2: A database as an alternative structure

Translate this query to logic, once with the classical vocabulary, and once with the alternative vocabulary. Which of those lies closer to the syntax of the SQL query?

Exercise 7. Suppose the table of Employees has a primary key, consisting of the first column. Which improvements can be made to the logical vocabulary and structure? (Hint: which property do the alternative vocabulary and a primary key have in common?)

[EXTRA] Exercise 8. Suppose a database with two tables *suppliers(supplier_id, supplier_name)* and *orders(supplier_id, quantity, price)*. The column *supplier_id* is a primary key in the first table.

Convert this database to a logical vocabulary and structure.

Using this vocabulary, convert this query to logic.

```

SELECT orders.quantity, orders.price
FROM orders
WHERE EXISTS(SELECT suppliers.supplier_id
FROM suppliers
WHERE suppliers.supplier_name = 'IBM'
and suppliers.supplier_id = orders.supplier_id);

```

3 A database as a logical theory

Aside from viewing a database as a structure, we can also take the viewpoint of seeing them as a logical theory, independent of a structure. This allows to do (for example) theorem proving.

A translation of a database to a theory is successful if a theorem prover can answer all our queries with the same answer as a database system.

What is the purpose of this? Do we mean that database systems should be replaced with theorem provers? That would be ridiculous. No, we rather

mean that a database system can -also- be regarded as a super efficient theorem prover for a very specific and simple kind of information, the kind of information we find inside of databases. We simply want to understand which kinds of information can be represented in a database and which cannot. We will notice that there are a few important and unexpected lessons to learn about the modelling of information in logic.

Exercise 9. Construct a theory T_{DB} which expresses exactly the information of the student database. Do not use constructed types, use constants to represent the students, courses, grades and teachers.

Exercise 10. The following queries are true in the database, are they also true in the theory you just constructed? Prove that they are true or give a structure in which they aren't. (You don't need to improve your theory if some queries fail, this will be assessed later in this assignment)

- $\neg Ray = Hec$
- $\neg Instructor(Ray, CS238)$
- $\neg(\exists x) Prerequ(x, x)$ - an integrity constraint
- $(\forall x)(Prerequ(x, CS238) \Rightarrow Instructor(Ray, x))$

If we look to the queries of the student database, we can prove that no query that contains $\Rightarrow, \neq, \Leftrightarrow$ or \forall is provable from T_{DB} . Only the most simple true queries can be answered with a theorem prover starting from T_{DB} .

Here we bump again into an important aspect of modelling: *implicit knowledge*. There are three kinds of implicit information in the database system in a database system which needs to be modelled explicitly in a logical theory (because logical theories give the user more freedom).

- In a database different strings point to different objects. So we know that $Ray, Hec, Sam, CS230, AAA, \dots$ are all different things. This was not expressed in T_{DB} , so the theorem prover couldn't know that $\neg Ray = Hec$. These inequalities are called the *Unique Name Axioms*.

Exercise 11. Express this information in logic

- We have not expressed that there are no other objects in the universe that are applicable, than the ones in our database domain:

$$\{Ray, \dots, Sam, \dots, CS230, \dots, AAA, \dots\}.$$

The axiom to express this, is called the *Domain Closure Axiom*.

Exercise 12. Express this information in logic

- Our logical theory contained the information which tuples were in the tables but not which tuples were not in the tables. For example: the theory does not express that $\neg \text{Prerequ}(CS230, CS230)$ so the theorem prover can not prove that no course is a prerequisite of itself. There are more tuples which are not in the table than those who are, and over those no information is present in our theory. To express exactly which tuples are present and which aren't, we need *definitions*.

Exercise 13. Express this information in logic

3.1 Null-values.

Suppose that in a table of the database, one or more values are lacking. In classical databases, one places NULL on those positions. One calls this a null-value. This means that the value for a position is unknown, for example:

Instructor	
Ray	NULL
Hec	CS230
Sue	NULL
NULL	M200
Pat	CS238

Intuitively this means the following: the table represents all information about who instructs which courses except we don't know which course is instructed by Ray and we don't know who instructs M200. Note that it is possible that Ray instructs M200.

Exercise 14. Modify the logical theory so it contains all information from the *Instructor* table.

Exercise 15. Suppose we want the additional constraint: the unknown courses that Ray and Sue instruct are different. Express this in your solution.

Exercise 16. Suppose we want the additional constraint: The unknown course given by Ray is either CS230 or CS 238. Express this in your solution.