Carleton
UNIVERSITY

Database Management Systems

Prof. Leo Bertossi

School of Computer Science

Carleton University

Ottawa, Canada

bertossi@scs.carleton.ca

www.scs.carleton.ca/~bertossi

Chapter 1: Introduction

Data Models and Database Systems

Data Models

There is data out there ...

If we want to understand, manipulate, transform, update, extract, process it, we need to create the right models of data

What is a model?

Basically, an abstraction, a simplified description or representation of a certain reality, phenomenon, company, etc.

From the model its is possible to extract both explicit and implicit information

We have found mathematical models in many situations:

- A mathematical model of a continuous physical phenomenon is usually given using the language of differential equations

  Variables representing different physical magnitudes are introduced and related by means of equations

  Solving the equations allows us to predict values for the system being described

- In operational research we find models that are expressed using linear algebraic (in)equations

  Using the methods of linear or matrix algebra we can obtain the values for the variables involved under certain conditions

  Again the equations relate the different variables

- Models of electrical circuits are expressed using the language of Boolean algebra
- Many discrete phenomena can be modelled using the language of graphs
- We can use languages of symbolic logic to create models and represent knowledge

  In particular, a mathematical theory can be seen as a model

  The explicit knowledge are the axioms and the implicit knowledge are the theorems, that are obtained by logical inference

  A language of "predicate logic" can be used for that

There is no universal modeling language

If we want to model data, we usually create mathematical models of data

There are different ways of modeling data depending on the application and the (mathematical) elements we use to represent data

Some models are more suitable than others for certain situations and applications

Data is usually organized in a natural way

Data is also associated to other things

It is not just a chaotic bunch of numbers or alphanumeric strings

A model of data has to capture the items to which data is associated, the kinds of association, the dependencies between data items, the natural organization (if any), etc.

In the case of data, we usually find data items and relationships between data items

Also the kinds of data items involved are relevant

Example:

"*Peter spends $23 on a CD at CDWarehouse*"

"*Mary spends $30 on a book at Chapters*"

Here "Peter" is a data item, the same applies to "$23", "$30", "book", "Chapters", ...

Not only that: "Peter" is of the kind, say "Customer", "CD" of the kind "Article", ...

We have categories of data items or concepts, entities

Even more, data item "Peter" is related to data item "CD", but not to "book", etc.

How can we capture this?

Notice that this data set is (seems to be) quite "structured"

What about this representation?

Sales	Customer	Price	Article	Store
	peter	$23	cd	cdWarehouse
	mary	$30	book	chapters

It is indeed a simplified representation

It captures the relationships between data items

It somehow captures the fact that data items are of different kinds

Is this a mathematical model? It looks just like a table ...

It can be reduced to a mathematical model, actually "set-theoretic model", consisting of:

- An underlying domain of data objects (or data items)

  $U = \{peter, mary, \$23, \$30, cd, book, cdWarehouse, chapters, john, ...\}$ (possibly infinite)

- Unary predicates over that domain, aka "Attributes":

  $Customer(\cdot), Price(\cdot), Article(\cdot), Store(\cdot)$

  Unary predicates are properties that apply to one individual at a time, e.g. $IsRead(\cdot)$, $IsKevinsUncle(\cdot)$

  Binary predicates are properties of two individuals, e.g. $BeingUncleOf(\cdot, \cdot)$, etc.

These attribute predicates have extensions:

Ext. of $Customer = \{peter, mary, \ldots\}$, etc.

That is, the property of being a customer applies to Peter, usually denoted $Customer(peter)$

(In the other examples, we may have $IsKevinsUncle(john)$, but possibly not $IsKevinsUncle(mary)$

Similarly, $BeingUncleOf(john, kevin)$)

Notice that for the extensions it holds

Ext. of $Customer$, Ext. of $Price$, etc. $\subseteq U$

Each of the attributes has a (possibly infinite) subdomain contained in U, e.g. $Dom(Customer) \subseteq U$

- A relational predicate $Sales(\cdot, \cdot, \cdot, \cdot)$, a 4-ary predicate

  This predicate is a *name* for a property that applies to 4 individuals ate a time

  A relation (or instance) for this predicate is an *extension*, i.e. a set of 4-tuples

  A finite relation in the case of relational DBs

  Here, Ext. of $Sales = \{(peter, \$23, cd, cdWarehouse), \ldots\}$

  Notice that for the extension it holds

  Ext. of $Sales \subseteq U^4 \quad (:= U \times U \times U \times U)$

  We can be even more precise:

  Ext. of $Sales \subseteq Dom(Customer) \times Dom(Price) \times Dom(Article) \times Dom(Store)$

The domains on the RHS can be infinite, but the extension of $Sales$ is finite (a reason for identifying it with a table)

So, the whole model can be represented in set-theoretic terms

This is an example of a relational model of data

The relational model of data and its computational representation and processing through relational database management systems (DBMSs) have been together one of the most striking conceptual, technical and commercial success stories in computer science!

- Data is organized and represented in terms of relations
- Data is processed by appealing to set-theoretic operations on relations

  There is a set-theoretic algebra of relations that can be put to good use

  can use algebra to manipulate sets
- The languages of predicate logic can be used to express many things in the model, e.g. queries

A powerful and mathematically clean combination of set theory and symbolic logic!

Example: (informal for now, do not worry ...) A possible query expressed in natural language

Give me the names of the customers who have bought an article for more than $30 at Chapters

This cannot be understood by the database management system (DBMS)

The query has to be expressed in a language that if machine processable and compatible with the database stored in the DBMS language has to have some primitives that allows us to connect to the database

A language of predicate logic is adequate for that

Let's see ...

The table provides a clear and nice "logical view of data"

The user should be confronted with that logical view; without having to care much about how the material, physical data is really stored in the computer, by means of what data structures or access methods

The relational model provides such logical view of data

in the example, we have:

the schema, that captures the structure of the data, and specifies the domain, relation names (database predicates), attributes extensions etc.

Supply	Company	Receiver	Item
	C	D1	I1
	D	D2	I2

Articles	Item	Class
	I1	K
	I2	K

2 x 3 table for supply and 2 x 2 table for articles

- schema containing an underlying domain, two relations of arity 3 and 2; with 3 and 2 attributes, resp. ; 4 attributes (one shared by two relations)

- Whenever we describe the schema, we should talk about predicates rather than relations: the latter are the extensions of the former. however, we usually do not make the distinction)

- in addition to the schema, we have the extensions shown in the tables

1. First of all, we are asking about tuples (rows) in the relation above (the table)
   Initially, they are, generically speaking of the form

$$Sales(x, y, z, w),$$

4 values that satisfy the predicate and/or make the predicate true

   where x is a variable that stands for a customer, etc.
2. We want the price to be greater than $30, we impose this condition

$$Sales(x, y, z, w) \wedge y > 30$$

predicate of what arity?
- binary

3. And the store to be Chapters:

$$Sales(x, y, z, w) \wedge y > 30 \wedge w = chapters$$

collecting 4 values in total; x,y,z,w

4. We are interested only in the customer names, so we collect those for which there are values for y, z, w in the tuple for which the conditions above hold, but we do not care about those values

these values are bound, however, x is free

Just collecting the x $Q(x)$: $\exists y \exists z \exists w (Sales(x, y, z, w) \wedge y > 30 \wedge w = chapters)$

if you start quantifying x the meaning of the condition changes completely… not going to collect any values. If you add an Exist for x as well it becomes a boolean or a yes/no query

This is the final query; it defines a new (unary) predicate
The values that make it true are the answers to the query

what kind of query is this?
- declarative query
  -- expressing what we want from the database, but we do not tell the computer or the database how to compute those values