

# Advanced Data Management

## Course Introduction

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Introduction

# Course Objectives

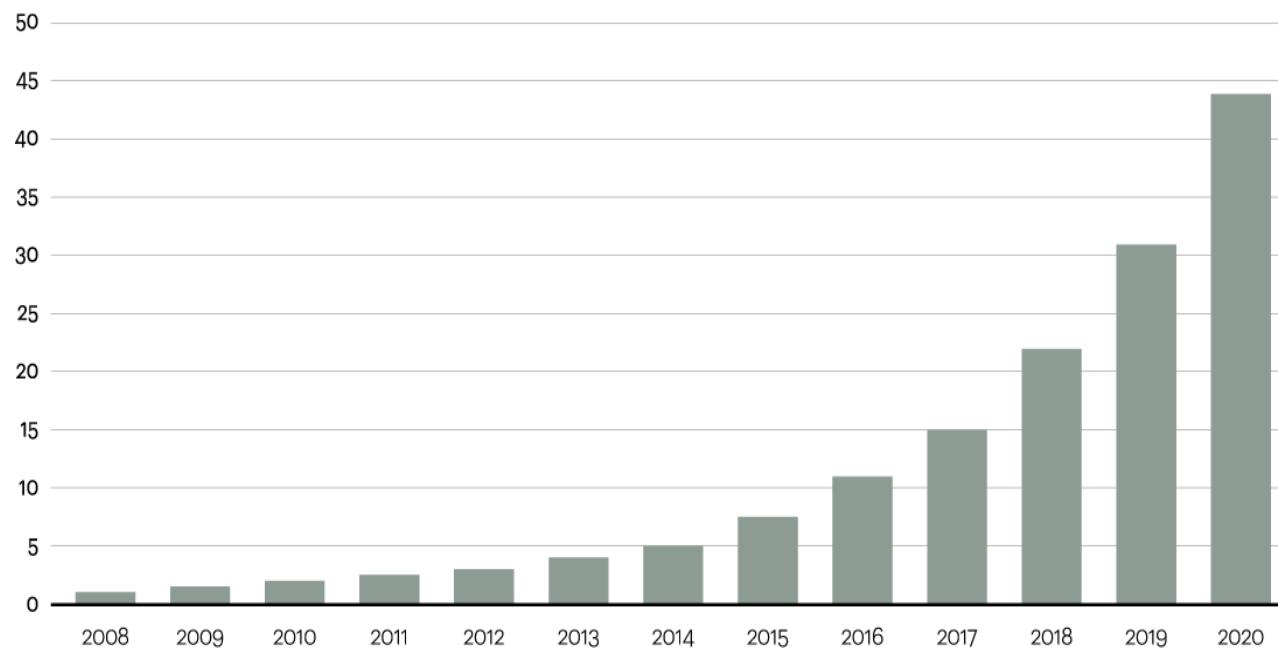
1. Provide the basics of information integration
2. Develop a rigorous theory of data integration through logics

**Why?**

# Data is Ever-Growing!

**Data is growing at a 40 percent compound annual rate, reaching nearly 45 ZB by 2020**

**Data in zettabytes (ZB)**



Source: Oracle, 2012

Source: <http://www.atkearney.com/>

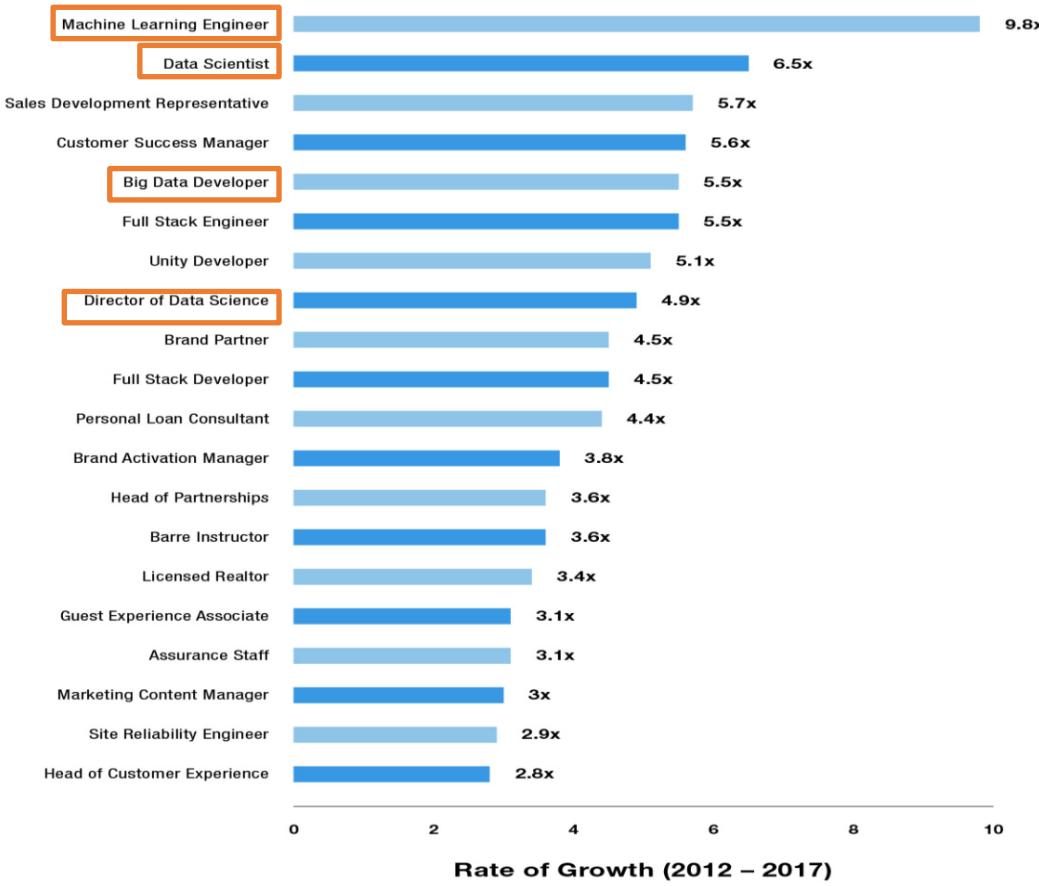
## WHAT'S A ZETTABYTE?

1 kilobyte	1,000,000,000,000,000,000
1 megabyte	1,000,000,000,000,000,000
1 gigabyte	1,000,000,000,000,000,000
1 terabyte	1,000,000,000,000,000,000
1 petabyte	1,000,000,000,000,000,000
1 exabyte	1,000,000,000,000,000,000
1 zettabyte	1,000,000,000,000,000,000

SOURCES: CISCO

# The Data Market is Ever-Growing

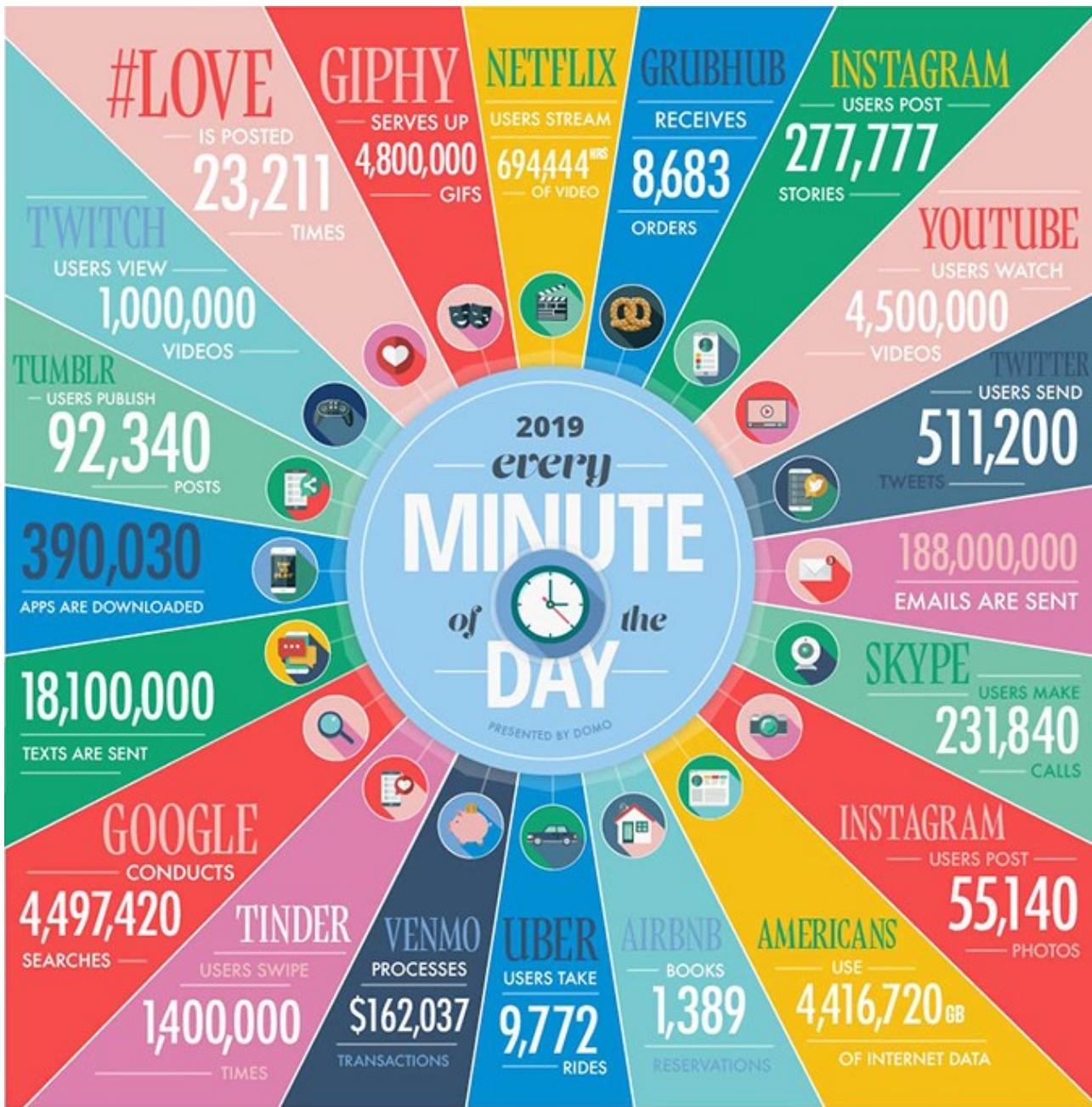
## Top 20 Emerging Jobs



# **Who is producing data**

Who is producing all this data?

1. Business Information Systems
2. Social media
3. Sensors and IoT
- 4. US!**



# Pervasive Computing

Lots of data is produced by machines



**SMARTWATCHES**

**DOMOTICS**

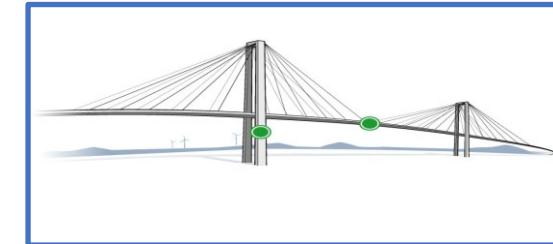


**CARS**



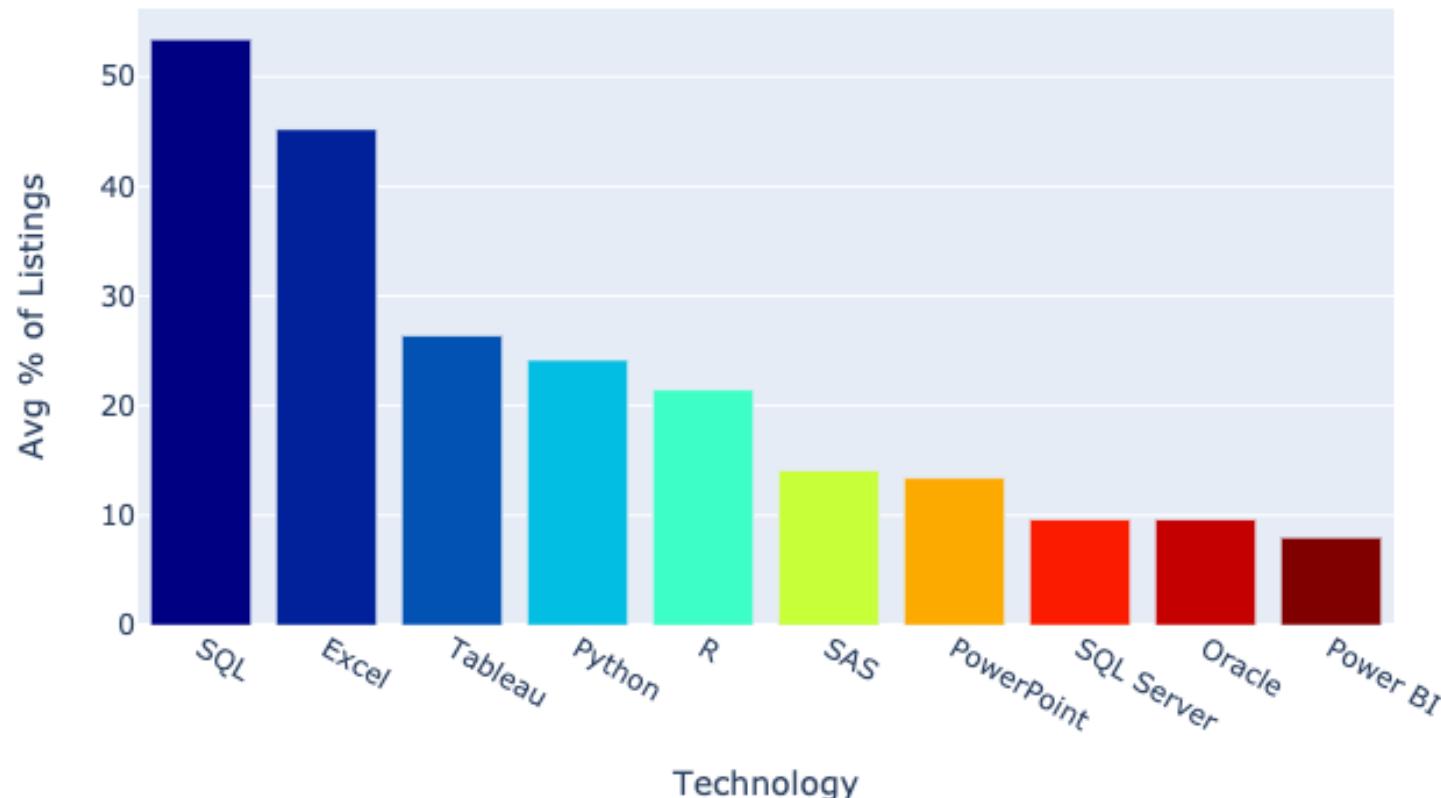
**SMART LIGHTNING SYSTEM**

**STRUCTURAL MONITORING**



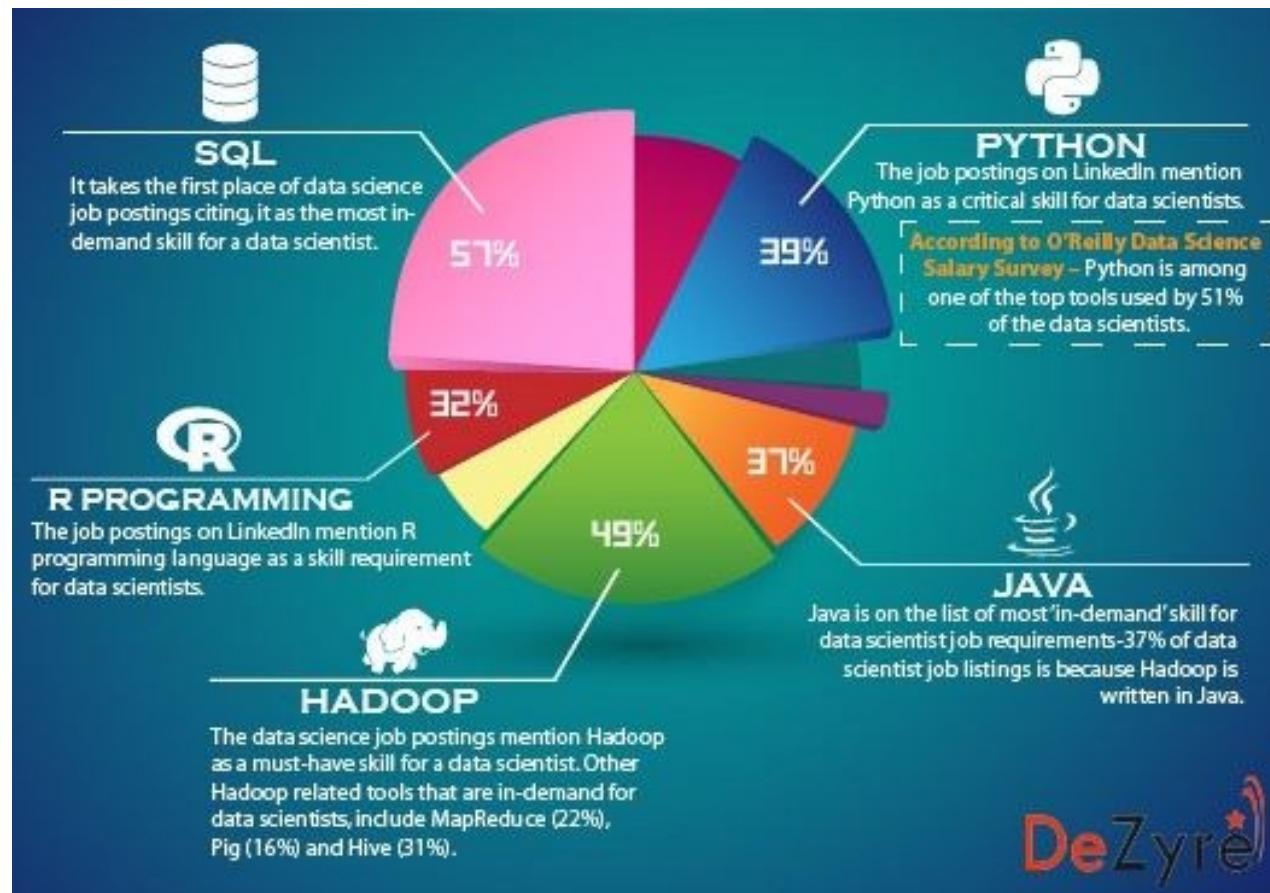
# Structured Data Manipulation!

Technologies in Data Analyst Job Listings 2020



# Structured Data Manipulation!

## Data Scientist Skills in Job Ads



# **Structured Data Manipulation!**

**Why structured data manipulation is so important?**

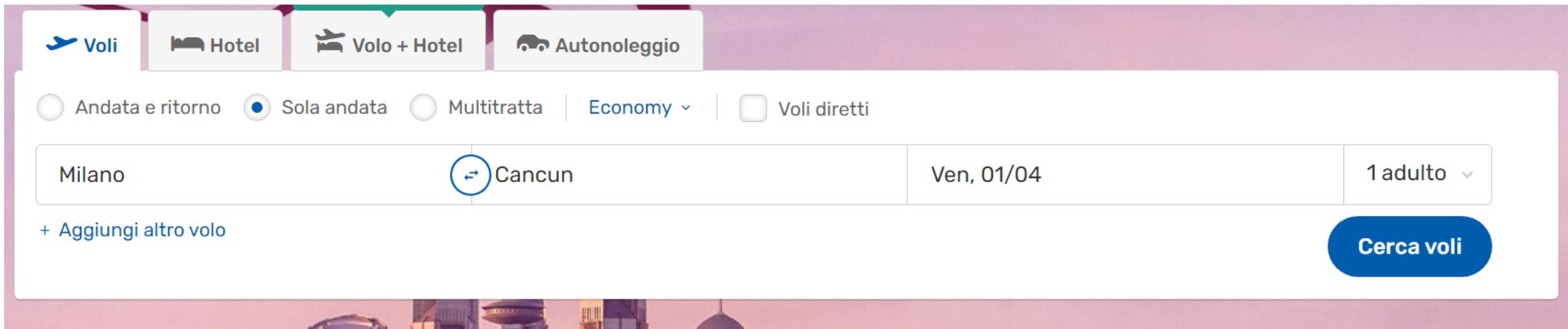
# **Structured Data Manipulation!**

It allows companies and organizations to **effectively access their data** and commutate it into the **information** that is **critical to their business and aims**.

Structured data manipulation process is also a valuable tool in **identifying data inconsistencies and redundancies**.

Enable **valuable functionalities and services** for companies and their customers\users.

# Example: Escape the Advanced Data Management Course!



1. Open the list of destinations (Destinations available from different websites)
2. Open the list of available dates (Dates available from different websites)
3. Search...

...Where does the data come from?

# Example: Escape the Advanced Data Management Course!

- Where do flight options come from?
  - Different sources: companies, brokers...
  - Different entities may represent data (flights) in different ways
- Are the flight options still available?
  - Your colleagues may have booked all the flights already!
  - Data may have been pre-loaded and not up to date.
- Confirm booking.
  - The escape is set!
  - You produce more data (your seat is no longer available).

# Example: Escape the Advanced Data Management Course!

- Where do flight options come from?
  - Different sources: companies, brokers...
  - Different entities may represent data (flights) in different ways
- Are the flight options correct?
  - Your flight is delayed
  - Data is incomplete
- Confirm the escape
  - The escape is set!
  - You produce more data (your seat is no longer available).

We need integrating information  
from multiple sources

# Integrating Information from Multiple Sources

- Integrating information from multiple sources is **challenging**
- Data may reside in **different physical** places
  - Load a snapshot? (**materialization**)
  - Keep hitting the sources with queries? (**virtualization**)
- Different sources may represent data in different ways (**heterogeneity**)
  - Physically, data may be stored in different formats
  - Conceptually, data may represent different aspects of the same reality.

The goal of information integration is to provide the user with a reconciled view of data stored in different sources

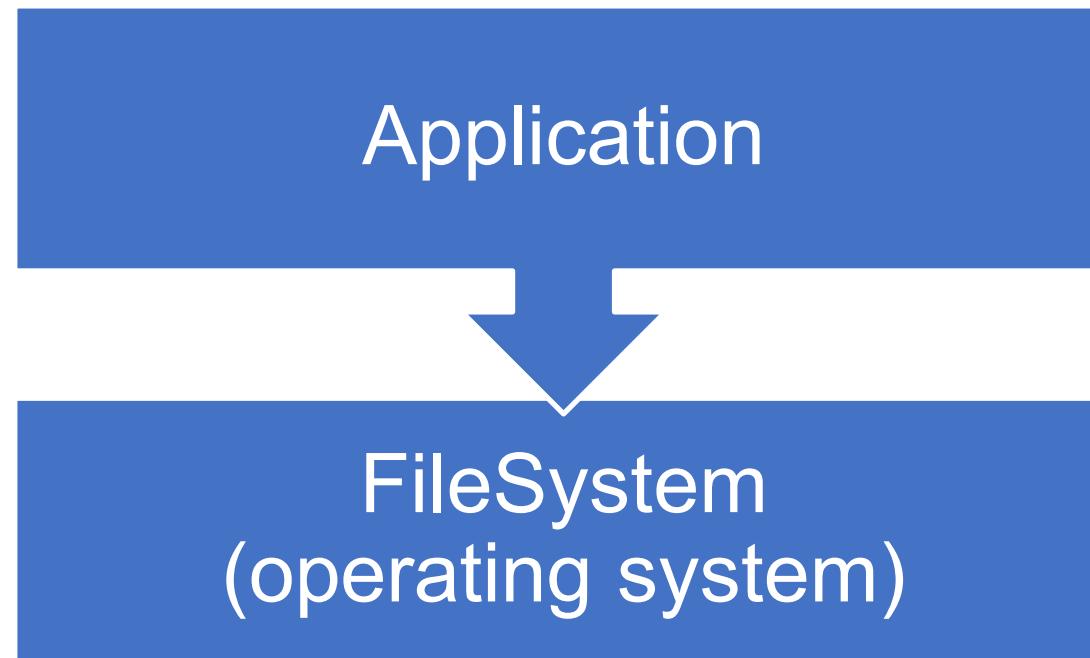
# Data Heterogeneity

# **Physical Heterogeneity**

- Information may be stored in different formats
  - Depending on the technology used

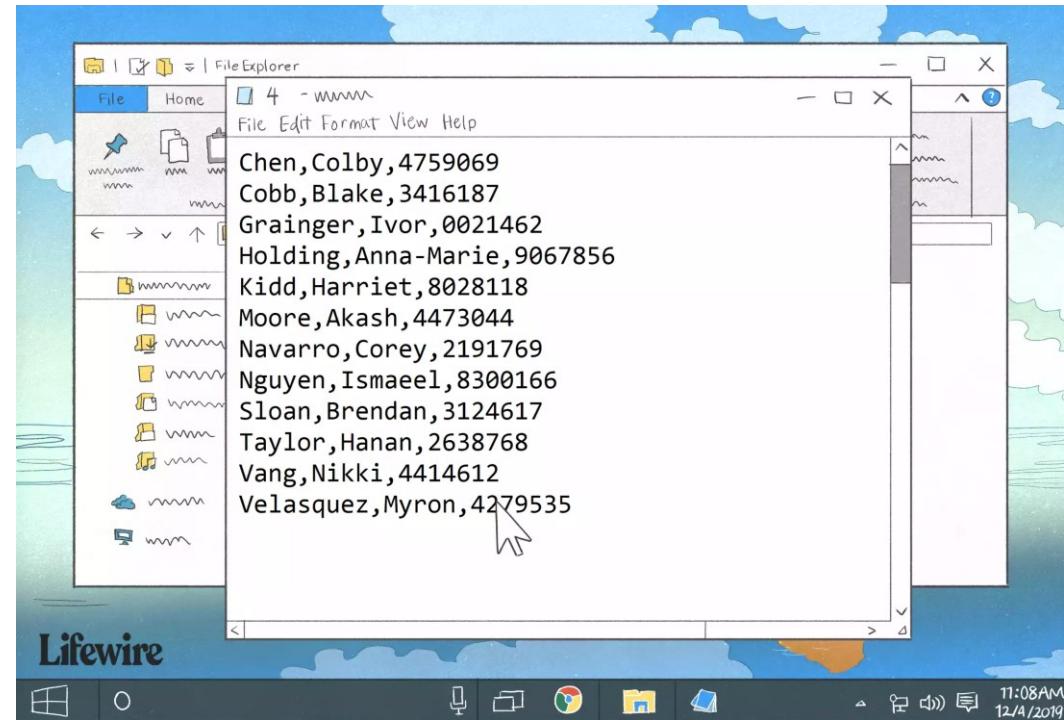
# File-Based Approach

- Applications manage information using the OS file system via APIs



# File-Based Approach

- The information in the files may take different shapes
- **Structured.**
  - Information takes a tabular shape.
  - **Example.** Comma Separated Values (CSV) files



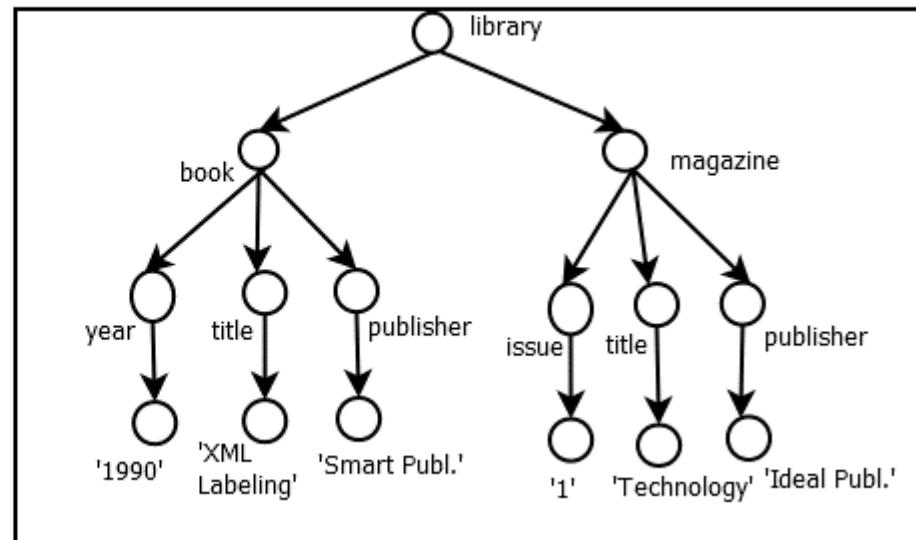
# File-Based Approach

- The information in the files may take different shapes
- **Semi-Structured.**
  - Information has a shape, but there is more flexibility.
  - **Example.** eXtensible Markup Language (XML) files

a) XML document

```
<library>
  <book>
    <year>1990</year>
    <title>XML Labeling</title>
    <publisher>Smart Publ.</publisher>
  </book>
  <magazine>
    <issue>1</issue>
    <title>Technology</title>
    <publisher>Ideal Publ.</publisher>
  </magazine>
</library>
```

b) XML Tree representation



# File-Based Approach

- The information in the files may take different shapes
- **Unstructured.**
  - Information has no predefined structure.
  - **Example.** Text (TXT) files
  - Hard to interpret...



This image displays a large block of random, illegible text. The characters are a mix of uppercase and lowercase letters, numbers, and symbols, arranged in a grid-like pattern. There is no discernible structure or meaning to the text, which serves as a visual representation of unstructured data.

# File-Based Approach

- The information in the files may take different shapes
- **Unstructured.**
  - Information has no predefined structure.
  - **Example.** Text (TXT) files
  - Hard to interpret...

```
:7c7irrJUhf1tfjfUft1Ucx0r: RL
rULiiij7JF1hffUFjJL7YcjJYrMXhRD.MY
:fTi::ct1ftUtjtJJ77r7ctr7r7riMPjtFQ.OU
itj:.LffUtJUYJJJL7rcUDebptiii:iQU:Y0dc F1
7FY,:YhffJjcY77r77r:RQD0:.7L:::LQr,:i7jJ D3
:EL:.t12jUYJ77rLtX7i:r9.R:7Uzr...:hQ...:iLjY Ri
:2,:UsffjjcYrrjPhb5J..YJ Qf51R...:QX...:::7JL Q.
,JFFUtljcLr2QRF..Y..:D :R2fXX...:c7.:::::rcr Q r9.cDh2jfjtYURp7Jcc9tt79rQcJccLYj2hDZPi rDr
'h22utL7rr:.R. brfb...Z DpJtXJ...:PQ.::::::::::r7;.R rt fE22utUljcQ97cJLX52tp2 RUJj9Dfc7Jfp9Mxi 1Z
'ttJcrLFL:::E .R1X0.:::S Qjjc9...:QJ.:::::iCxr:Xi.X9ffjtJtjLLM27JYchXU29F QcJJ01R0h7cJ1hbDX. .Q:
ULr7990c..LP XE2bc..rX.QUUFZrcPORE. ,:7259b:,pXUjJjjtJUYJ7REr7JLP5tbbF QcLjXLr9QQjYYUFxOML RJ
LrF07:ci..X. Q229: t:tQQQQQQMMZQQQJ:irrJ2PhFfpZbRRbp12jJJUJLr:YQY7:::JSUFPX Q7cY0jjp cQ7JYJJ15ZF, +
:LOXijD.::Q .QUTX_LDQ , iERQQZ0hXF2LY2bftpZ9PSS1XSSUjtU:::rJYSMQQMjhB2JPS RLLjpffP QjYYLcYf2E0i
,S3MUP5..:Q DQRQQQQJ .QQDUjJFh99bFL7NMbjJYUUFFFhj7rPPRQE9RQQQQQQQ Q7YrpU1Q QTjZpjL29EL
.bF9UpY..7R bQD7. OQQFr::i7JUhfPxDQt7c7L7ccftF29XrrQMDMDQQQLr,.LJ..QiicPfjQ QYcffUZ17tfP2
:QEHjE..rQ. QQRp7: irJU2t0QJ7Lrr.:iLLfjf2Pb:::QQQML.7. MQE:pJUQ QLjY07LR2Luf)
JtRjh0YQQ: QQ99jYr:::::7YfjsQErJL7i..i7YJtU119P 9r.rQ., 0: .1QQQOQ ,QL73ptUb2fLut
D7QQQQP. QQFP1jJjL7rcJttfbQj7jtLL777cYUJffFSZ.7r 7R.ii,R. ,FQQ QLLjpjhJfcLJ
Q,1t. cEXU1FUtjUjtft21MQjrfFFjtJUJUjttF1pM:7E79J,:r:Q .Q:rtpthjUFLJL
jh jLD07hF1jtj2U1fhfhoQh725ffUtjfU2fh159Z PhiMr ..r.Yh bQL:pJhtifcJ
: tF2DQLJXhh212h1h1pEQ7jhXF2U2t12h1xpR: QQS:t tr.i.R. 7QDRt2222LjJ
:pjD1EDLjPxphXFShXFf2RQRLtSPFSFSFSh9bRi QRMRr:i99::i:R JQQQXStc7]
,LL7hjQZQQZQQ0t115F52Ft1SRRpYRLLfPXPSPPZ21, RQDQbL,1tJt.:,PL i0.P11L7
:Yp1fii.. .i5QQQQP5hPPZQRQEY:XQ9Jctfh1tr.,7QQRDRME.rUQQj,:::Q MS7cJ
,rEti ,:::rri:. 19QQQhRRREMQQME0QQR9XjLr7jQQRDRRBY,t7QQ. .:D: QR:r
20r ,7ftUyjpoE9Pt7 rRR10DRRQjr:icPMR0QQMQRMDRRBt.57URR :::rQ hQl
```

# File-Based Approach

- The information in the files may take different shapes
  - **Unstructured.**
    - Information has no predefined structure.
    - **Example.** Text (TXT) files
    - Hard to interpret...

# **File-Based Storage: Examples**

1. XML Files
2. Excel sheets
3. CSV files.
4. Text

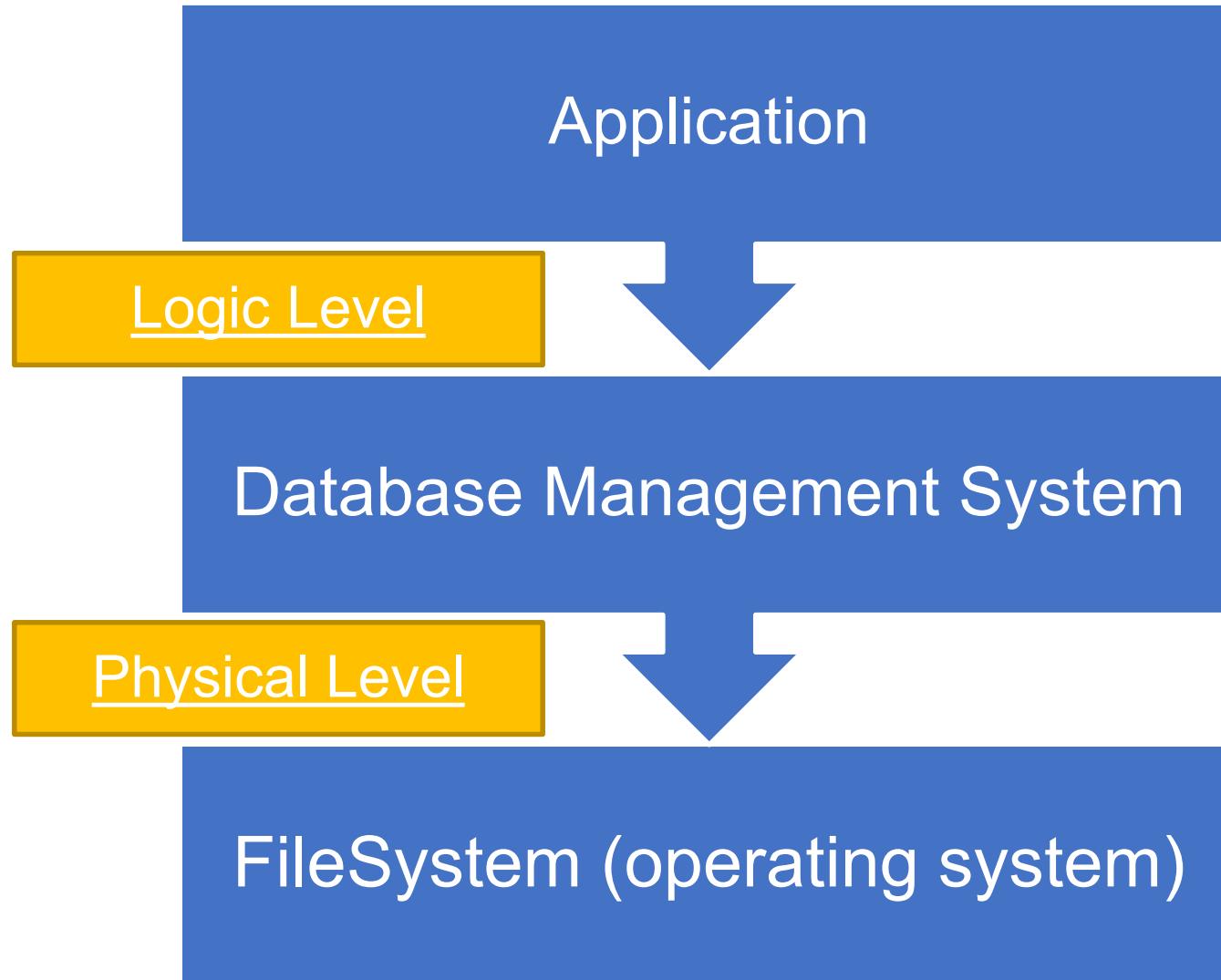
## A More Structured Approach.

- In many cases, we cannot rely only on files.
- If the size of the data is big (GBs,TBs...)
  - We need efficient mechanisms to store and access
  - Implementing the right algorithm to navigate the data in our application may not be easy
- If multiple concurrent accesses are possible
  - We need to manage concurrent updates
  - Algorithms for that may be tricky to implement.
- In all these cases, the management of data should be left to a specialized piece of software

# Database Management Systems

- In these cases we use **Data**Base** **M**anagement **S**ystem**
  - Specialized piece of software that handles data
- Using DBMSs, the application and data layer are separate
  - Applications interact with the data at the logical level while the physical level is handled by the DBMS
- DBMSs do not solve the information integration problem
  - Different DBMSs may use different data model
  - Different DBMSs may represent data in different way

# Application logic / data separation



# Different Data Models

- **Legacy databases**
  - Hierarchical model
  - Network model
- **Post-Relational**
  - Object Databases
  - Multivalue DBs
  - Graphs
- **SQL database.**
  - We can use SQL or a dialect thereof.
  - Relational Data Model.
- **NoSQL Databases**
  - Key-value stores
  - Big Tables
  - Documents (Json or the like)

# Conceptual Heterogeneity

Information may be represented in different ways from the conceptual point of view.

# Conceptual Heterogeneity: Example

Name	DOB	Residency
Amber A.	10/20/2030	New Amsterdam
Rose B.	10/20/1930	Constantinople

**Both databases are relational.**

- What are the persons?
- What's their DOB?

ID	Customer Name	Product
O001	Amber A.	P001
O002	Dylan B.	P001

# Conceptual Heterogeneity: Example

Person	DOB
Amber A	10/20/2030
Rose B.	10/20/1930
Dylan B.	Unknown

# Conceptual Heterogeneity: Example

Person	DOB
Amber A	10/20/2030
Rose B.	10/20/1930
Dylan B.	10/20/2030

# Conceptual Heterogeneity: Example

Person	DOB
Amber A.	10/20/2030
Amber A.	<b>Unknown</b>
Rose B.	10/20/1930
Dylan B.	<b>Unknown</b>

# Conceptual Heterogeneity: Example

Person	DOB
Amber A.	10/20/2030
Rose B.	10/20/1930
Dylan B.	<b>Unknown</b>
Abraham L.	10/20/1930

# Information Management

# Data Management

We may want to perform different tasks with the reconciled information:

- **Query:** Answer database queries over the different sources
- **Update:** Change information at the sources
- **Delete:** Remove information from the sources
- **Insert:** Add information into the sources

# Source

Name	DOB	Residency
Amber A.	10/20/2030	New Amsterdam
Rose B.	10/20/1930	Constantinople

ID	Customer Name	Product
O001	Amber A.	P001
O002	Dylan B.	P001

# Query

- How do we extract information from the sources?
- **Ex: Extract all persons in the data with their Date of Birth (if known).**

# Materialization

Person	DOB
Amber A.	10/20/2030
Rose B.	10/20/1930
Dylan B.	Unknown

# Virtualized

Name	DOB	Residency
Amber A.	10/20/2030	New Amsterdam
Rose B.	10/20/1930	Constantinople

ID	CName	Product
O001	Amber A.	P001
O002	Dylan B.	P001

```
SELECT Name, DOB  
FROM Persons
```

```
SELECT CName  
FROM Customers
```

# Virtual vs Materialized

- **Virtual PROs.**
  - Information is always fresh.
  - No need for extra space.
- **Virtual CONs.**
  - Usually, computationally heavy.
  - Multiple accesses to sources
- **Materialized PROs.**
  - Efficiency (after materialization is done).
  - Single access to the data sources.
- **Materialized CONs.**
  - Requires extra space.
  - Data may not be up-to-date.

## Other Data Management Tasks

- How do we update, delete, or insert information in the system?
- **Ex: Add a person in the data.**

# Materialization

Person	DOB
Amber A.	10/20/2030
Rose B.	10/20/1930
Dylan B.	Unknown
Deckard R.	Unknown

# Virtualized

Name	DOB	Residency
Amber A.	10/20/2030	New Amsterdam
Rose B.	10/20/1930	Constantinople

ID	CName	Product
O001	Amber A.	P001
O002	Dylan B.	P001

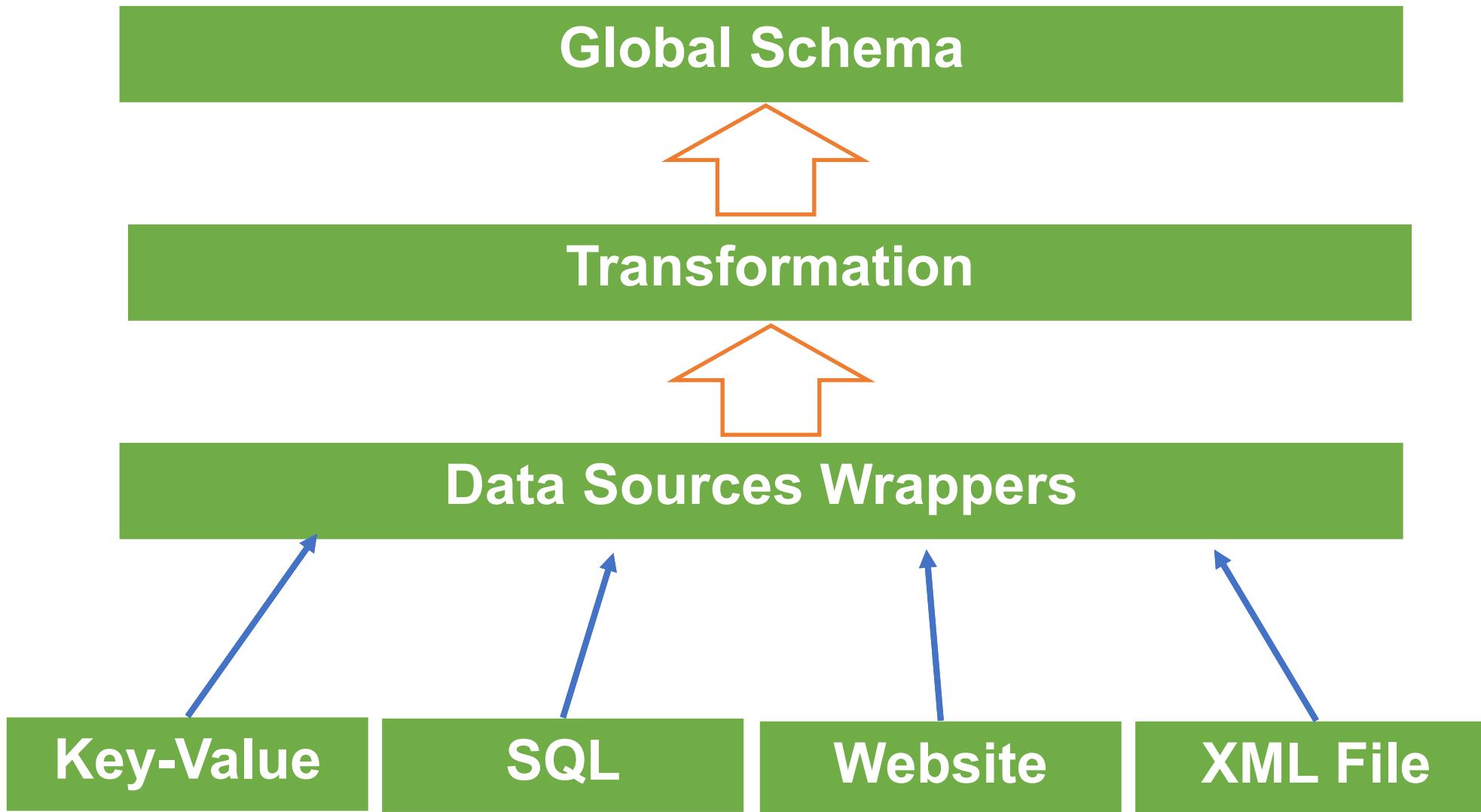
Deckard R.		
------------	--	--

# Solutions?

# Dealing with Heterogeneity

- Physical Heterogeneity: **Wrappers**, i.e., pieces of software that extract data from the sources.
- Conceptual Heterogeneity: **Global Schema**, i.e., a conceptual description of the information we need.
- Result of the Integration: transformation from data at the sources to data for the global schema (mapping).

# Dealing with Heterogeneity



# Information Integration

**Information integration** is the problem of providing unified and transparent access to a collection of data stored in **multiple**, **autonomous**, and **heterogeneous** data sources.

Two different classes of approaches:

- **Materialization-based approaches**
  - A (small) representation of the result of the integration is loaded
  - Operations are carried out over the materialization
- **Virtualization-Based Approaches**
  - Tasks for the global schema are translated as queries for the data sources.

In both cases we need to define what the integration process means.

- We need to provide **semantics** for the specification

# Challenges Ahead

- Information integration is (not only) a technological problem.
- We need to develop a **theory** to accommodate all its aspects.
  - How do we define a global schema?
  - How mapping reconcile the global schema with the sources?
  - How to answer queries expressed on the global schema?

In this part of the course, we aim to introduce the problem of **data integration** from a formal point of view and to develop such theory relying on formal tools such as

- The relational model of data
- Logic (Propositional Logic, First-Order Logic, and Description Logics)

# Outline of the course

1. Introduction to Propositional Logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. **Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Propositional Logic

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. **Introduction to Propositional Logic**
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. **Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# **Formal Logic**

# Formal Logic

When dealing with important things in life (...like databases...) we need to agree on a way to evaluate our sentences.

- Is a database correctly representing our scenario?
- Is a tuple satisfying a given condition?

To do that, in Computer Science and Engineering we borrow constructs from a branch of mathematics and philosophy called **formal logic**.

## Formal Logic:

- The **formal** study of human reasoning (**Philosophy**).
- The use of logical formalisms to study mathematical objects (**Math**).

# Example

On a walk in the woods, you stumble upon a bridge guarded by two trolls (named **Ander** and **Bjorn**). They say:

- **Ander**: If one of us is a knight, then you can safely cross the bridge.
- **Bjorn**: Either Ander is a knight or Bjorn is a knight.

Yes... They refer to themselves in the third person... Trolls...

Questions:

1. *If both trolls are telling the truth, is it safe to cross the bridge?*
2. *If one troll lies, can we conclude that it is not safe to cross the bridge?*
3. *Can they both lie?*

# Question 1

*If both trolls are telling the truth, is it safe to cross the bridge?*

**Ander:** If one of us is a knight then you can safely cross the bridge.

- If this sentence is true, to be sure we are safe we just need to find a knight!

**Bjorn:** Ander is a knight or Bjorn is a knight.

- If this sentence is true, we have our knight!
- **Conclusion:** you can safely cross!

## Question 2

*If one of the trolls lies, can we conclude that it is not safe to cross?*

- Suppose **Bjorn** lies:
  - **Bjorn** : Either Ander is a knight or Bjorn is a knight.
  - **Not Bjorn**: Ander is NOT a knight **AND** Bjorn is NOT a knight.
  - We cannot conclude that the bridge is safe to cross.
  - We cannot conclude that the bridge is not safe to cross.
  - **We do not know!**
- Suppose **Ander** lies.
  - **Ander**: If one of us is a knight, then you can safely cross the bridge.
  - **Not Ander**: One of us is a knight **AND** you can **NOT** safely cross the bridge.
  - We can conclude that it is **NOT** safe to cross!!
- If we know that Ander is lying, we can conclude that it is **not safe to cross!**

## Question 3

*Can both trolls lie?*

Suppose **Bjorn** lies:

- **Bjorn**: Either Ander is a knight or Bjorn is a knight.
- **Not Bjorn**: Ander is **NOT** a knight **AND** Bjorn is **NOT** a knight.

Suppose **Ander** lies:

- **Ander**: If one of us is a knight, then you can safely cross the bridge.
- **Not Ander**: One of us is a knight **AND** you can **NOT** safely cross the bridge.

So, we have that **One of them is a knight** (Ander) but **none of them is** (Bjorn).

→ **Contradiction!**

# Formal Logic: Some Examples

Formal logic can draw certain conclusions from certain premises.

- **Premise 1:** If Fabio is eating a croissant then Fabio is happy.
- **Premise 2:** Fabio is eating a croissant.
- **Conclusion:** Fabio is happy!

What did we just do? **Propositional Calculus!**

- We understood (calculated) whether a fact (proposition) is true (or false) starting from the fact that some other facts are certainly true (or false)
- Under the (somewhat hidden) assumption that fact is either **true** or **false**

# Truth Values

- What are the facts involved in our example?
  - Premise 1: If (Fabio is eating a croissant) then (Fabio is happy).
  - Premise 2: (Fabio is eating a croissant)
  - Conclusion: (Fabio is happy)
- At this level of abstraction, we only need to know if sentences are true
  - We call this the **truth value** of a sentence.
  - Unlike real life, we assume that every proposition is either **true** or **false**.
  - In the example, we concluded that (Fabio is Happy) is **true**.

# Propositional Variables

**Observe:** when dealing with truth values, we do not need to know exactly what the single fact is about. We just need to know its truth value.

For this reason, to keep notation compact, instead of writing down the whole proposition, we use **Propositional Variables**.

**A propositional variable is a symbol to which we associate a specific meaning.**

# Propositional Variables: Example

- Let's try to reason again about croissants using variables.

(Fabio is eating croissants) = C

(Fabio is happy) = H

- A straightforward translation.

Premise 1: If (Fabio is eating croissants) then (Fabio is happy).

- Premise 1: If C then H

Premise 2: (Fabio is eating a croissant)

- Premise 2: C

Conclusion: (Fabio is happy)

- Conclusion: H

# Formal Logic: Analyzing Connectives

When dealing with important things in life (...like databases and croissants...) we need to draw solid conclusions

- Premise 1: **If A then D**
- Premise 2: **(A AND B) OR ((NOT B) AND D)**
- Conclusion: **D is true?**

Evaluating complex examples requires a formal definition of how propositional variables are connected to each other.

**Propositional connectives: AND, OR, NOT ....**

# **Propositional Logic: Syntax**

# Alphabets, Symbols, and Words

- An **alphabet  $S$**  is simply a countably infinite set. The elements of  $S$  are called the **symbols** of the alphabet.
- A **word** (string)  $w$  over  $S$  is defined as follows:
  - Every symbol is a string.
  - If  $w, w'$  are strings, so is  $w \circ w'$
  - Nothing else is a word.
- Assume  $S = \{a, b\}$ 
  - Words:  $a, aa, ab, ba, aaa \dots$
  - The following is not a word of  $S$ :  $bac$

# Alphabet of Propositional Formulae

- Assume a countably infinite set  $R = \{p_1, p_2, \dots\}$ . ([Propositional Variables](#))
- Assume the set  $O = \{\neg, \wedge, \vee\}$  ([Connectives](#))
- Assume the set  $P$  of symbols “(” and “)”.
- Sets  $R$ ,  $O$ , and  $P$  form the **alphabet of propositional formulae**
- Not every word in this alphabet is a formula!

# Propositional Formulae

An (propositional) **atomic formula** is a (propositional) **variable**.

- For brevity, we often call atomic formulae simply **atoms**.

The set of propositional formulae  $\mathcal{L}$  is the smallest such that:

1. Every atom is in  $\mathcal{L}$
2. If  $f$  is in  $\mathcal{L}$ , then also  $\neg(f)$  is in  $\mathcal{L}$ .
3. If  $f, f'$  are in  $\mathcal{L}$ , so are  $(f \circ f')$ , with  $\circ \in \{\wedge, \vee\}$

## Propositional Formulae – Example

The following are propositional formulae

1.  $a$  (atom)
2.  $(a \wedge b)$
3.  $\neg((a \wedge b) \vee (c \wedge d))$

The following are not propositional formulae

1.  $a \wedge$
2.  $\vee b$
3.  $a \wedge b$  (no parenthesis, though sometimes we will omit them)

# **Propositional Logic: Semantics**

# Propositional Connectives

The set of truth values is the set {true, false}

To each propositional connective, we associate functions

- Unary: {true, false} → {true, false}
- Binary: {true, false} × {true, false} → {true, false}

Intuitively, these functions allow us to define the meaning of formulae.

- Usually, we define these functions using “truth tables”.

## AND : Truth Table

The connective **AND** requires that **both** operands must be true

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

## OR : Truth Table

The connective **OR** requires that **at least one** operand must be true

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

## NOT: Truth Table

The connective **NOT** requires that the operand **must be false**

NOT	
TRUE	FALSE
FALSE	TRUE

# Propositional Interpretations

In order to understand whether a formula is *true* or *false*, we need to evaluate its variables.

A (propositional) **interpretation** is a function  $I : \mathcal{L} \rightarrow \{\text{true}, \text{false}\}$  such that

- $I(a) \in \{\text{true}, \text{false}\}$ , for each atom  $a \in \mathcal{L}$
- $I(\neg f) = \text{NOT } I(f)$
- $I(f \wedge f') = \text{AND}(I(f), I(f'))$
- $I(f \vee f') = \text{OR}(I(f), I(f'))$

**Observe:** an interpretation is uniquely determined by its interpretation of atoms

**Observe:** the same holds for the truth value of a formula.

## Evaluation of Propositional Formulae: example

Consider an interpretation  $I$  such that:

$$I(A) = \text{true}, \quad I(B) = \text{false}, \quad I(C) = \text{true}$$

and the formula  $(A \wedge B) \vee ((\neg B) \wedge C)$ , we have:

- $\text{AND}(I(A), I(B)) = \text{false}$
- $\text{NOT}(I(B)) = \text{true}$
- $\text{AND}(\text{true}, I(C)) = \text{true}$
- $\text{OR}(\text{false}, \text{true})$  is **true**

# Exercise

Considering the interpretation I defined as follows:

$$I(A) = \text{true}, \quad I(B) = \text{false}, \quad I(C) = \text{true}, \quad I(D) = \text{false}$$

What is the truth value of the following propositional formulae?

1.  $(A \vee B) \wedge (\neg(C) \vee D)$
2.  $\neg(A \vee B) \vee \neg(C \vee D)$
3.  $(\neg(A) \wedge \neg(B)) \vee (C \vee D)$

# Exercise

Considering the interpretation I defined as follows:

$$I(A) = \text{true}, \quad I(B) = \text{false}, \quad I(C) = \text{true}, \quad I(D) = \text{false}$$

What is the truth value of the following propositional formulae?

1.  $(A \vee B) \wedge (\neg(C) \vee D)$       **false**

2.  $\neg(A \vee B) \vee \neg(C \vee D)$       **false**

3.  $(\neg(A) \wedge \neg(B)) \vee (C \vee D)$       **true**

# Logical Consequence

# Models and Consequences

We call a finite set of formulae a **theory**.

- Formulae in a theory are often called its **axioms**.

Assume an interpretations  $I$ , we say that

- $I$  is a **model** of a formula  $f$  if  $I(f) = \text{true}$  (written  $I \models f$ ).
  - $I$  is a **model** of a theory  $F$  if  $I(f) = \text{true}, \text{for every } f \in F$  (written  $I \models F$ ).
- 
- A formula  $f$  is the **consequence** of a theory  $G$  ( $G \models f$ ) if for every interpretation  $I$  that is a **model** of  $G$  we have that  $I$  is a **model** of  $f$
  - **Consequences** of a set of formulae  $A$  are often called the **theorems** of  $A$ .

# Exercise

Which of the following are logical consequences?

$$1. \{p, (\neg(p) \vee q)\} \models q$$

$$2. \{(p \vee q)\} \models q$$

$$3. \{(\neg(p) \wedge p)\} \models q$$

## Exercise: Solution (1)

$\{p, (\neg(p) \vee q)\} \models q$  **holds.**

Since:

1. In every model of the theory  $p$  is **true**
2. In order for  $(\neg(p) \vee q)$  to be **true** then,  $q$  needs to be **true**
3. So, every model of the theory  $\{p, (\neg(p) \vee q)\}$  is also a model of the formula  $q$

## Exercise: Solution (2)

$\{(p \vee q)\} \vDash q$  **does not hold.**

Indeed, the interpretations with  $I(p) = \text{true}$ ,  $I(q) = \text{false}$  are **models** of the theory.

## Exercise: Solution (3)

$\{(\neg(p) \wedge p)\} \models q$  **holds.**

Since:

1. The set has **no model**
2. If a set has **no models**, every formula **is a consequence by definition (ex falso sequitur quodlibet)**

## Exercise: Solution (recap)

- Which of the following are logical consequences?

1.  $\{p, (\neg(p) \vee q)\} \models q$

– **True**

2.  $\{p \vee q\} \models q$

– **False**

3.  $\{\neg(p) \wedge p\} \models q$

– **True**

# Logical Equivalences

# Logical Equivalence

- Assume an interpretation  $I$ 
  - $I$  is a **model** of a formula  $f$  if  $I(f) = \text{true}$  ( $I \models f$ ).
  - $I$  is a **model** of a theory  $F$  if  $I(f) = \text{true}$ , for every  $f \in F$ . ( $I \models F$ ).
- A formula  $f$  is **equivalent** to a formula  $f'$  ( $f \equiv f'$ ) if:
  - $I$  is a model of  $f$  if and only if  $I$  is a model of  $f'$
  - We extend this notion to theories in the natural way.
- There are several useful equivalences that one can prove.

# Implication

So far, we used **if ... then** (implication) in our sentences, but we do not have a connective for that. However, we do not need one.

Indeed, **If f then g** (*f implies g*) requires that **g** is *true* whenever **f** is *true* (if **f** is *false*, then we *don't care* about **g**) also written ( $f \rightarrow g$ )

- **How do we translate it in our formal logic?**

When is the sentence **if f then g** true?

- If **f** is **false** then **g** can be either **true** or **false**, we don't care.
- If **f** is **true**, then **g** must be **true**.

## Implication: Truth Table

<u>IMPLIES(A,B)</u>	B TRUE	B FALSE
A TRUE	TRUE	FALSE
A FALSE	TRUE	TRUE

# Exercise

As we said, all the logical connectives we need are **AND**, **OR**, and **NOT**.

**Claim:** *IMPLIES* ( $f, g$ ) is **equivalent** to **OR** (*NOT*( $f$ ),  $g$ )

$$(f \rightarrow g) \equiv (\neg(f) \vee g)$$

**Prove the claim above.**

## Exercise 5: Solution

- As we said, all the logical connectives we need are **AND**, **OR**, and **NOT**.
- **Claim:** *IMPLIES* ( $f, g$ ) is equivalent to *OR* (*NOT*( $f$ ),  $g$ )
- **Proof:** Truth table of **NOT**( $f$ ) **OR**  $g$ .

– $f = \text{true}, g = \text{true}$ .	$\text{NOT}(f) \text{ OR } g = \text{true}$
– $f = \text{true}, g = \text{false}$	$\text{NOT}(f) \text{ OR } g = \text{false}$
– $f = \text{false}, g = \text{true}$	$\text{NOT}(f) \text{ OR } g = \text{true}$
– $f = \text{false}, g = \text{false}$	$\text{NOT}(f) \text{ OR } g = \text{true}$

# Logical Implication: the two truth tables

<u>A IMPLIES B</u>	B TRUE	B FALSE	<u>NOT(A) OR B</u>	B TRUE	B FALSE
A TRUE	TRUE	FALSE	A TRUE	TRUE	FALSE
A FALSE	TRUE	TRUE	A FALSE	TRUE	TRUE

# De Morgan's Law and Double Negation

Important equivalences for handling negation. Assume formulae  $f, g$ .

- De Morgan's Law 1:  $\neg(f \wedge g) \equiv (\neg f \vee \neg g)$
  - De Morgan's Law 2:  $\neg(f \vee g) \equiv (\neg f \wedge \neg g)$
  - Double Negation:  $\neg\neg f \equiv f$
- 
- **Observe:** these are not axioms. We can prove them!

# Exercise

- Prove De Morgan's laws and the law of double negation.

# Exercise: Solution

- Prove De Morgan's laws and the law of double negation.
- De Morgan's Law 1:  $\neg(f \wedge g) \equiv (\neg f \vee \neg g)$
- **Proof:** We prove the claim showing the truth tables.

Truth table  $\neg(f \wedge g)$       Truth table  $(\neg f \vee \neg g)$

• <b>f = true, g = true.</b>	<b>false</b>	<b>false</b>
• <b>f = true, g = false</b>	<b>true</b>	<b>true</b>
• <b>f = false, g = true</b>	<b>true</b>	<b>true</b>
• <b>f = false, g = false</b>	<b>true</b>	<b>true</b>

## Exercise: Solution

- Prove De Morgan's laws and the law of double negation.
- Double Negation:  $\neg\neg f \equiv f$
- **Proof:** We prove the claim showing the truth tables.

Truth table  $f$

- $f = \text{false}$
- $f = \text{true}$

Truth table  $\neg\neg f$

<b>false</b>	<b>false</b>
<b>true</b>	<b>true</b>

## Exercise (the trolls...again)

On a walk in the woods, you stumble upon a bridge guarded by two trolls. They say:

- **Ander**: If one of us is a knight, then you can safely cross the bridge.
  - **Bjorn**: Either Ander is a knight or Bjorn is a knight.
1. Prove, by means of propositional logic, that it is safe to cross the bridge.
  2. Prove, by means of propositional logic, that at least one of the two trolls tells the truth.

## Exercise: Solution (1)

(one knight) =  $K$ , (safe bridge) =  $S$

(Ander knight) =  $KA$ , (Bjorn knight) =  $KB$ ,

**Ander:** If one of us is a knight, then you can safely cross the bridge.

$\neg K \vee S$  (If K then S)

**Bjorn:** Either Ander is a knight or Bjorn is a knight.

$KA \vee KB$

Ander or Bjorn are knights if and only if there is a knight

$\neg(KA \vee KB) \vee K$

$\neg K \vee (KA \vee KB)$

## Exercise: Solution (2)

Consider the following theory

$$\mathcal{A} = \{(\neg K \vee S), (KA \vee KB), (\neg(KA \vee KB) \vee K), (\neg K \vee (KA \vee KB))\}$$

**To prove that it is safe to cross the bridge, we need to prove that  $\mathcal{A} \models S$  holds**

1. In each model of  $\mathcal{A}$ , it holds that  $(KA \vee KB)$ 
  - because in  $\mathcal{A}$  we have  $(KA \vee KB)$
2. Therefore,  $K$  holds
  - because  $(\neg(KA \vee KB) \vee K)$
3. Therefore  $S$  is true
  - because  $(\neg K \vee S)$
4. So, every model of  $\mathcal{A}$  is also a model for  $S$ , i.e.,  $\mathcal{A} \models S$

## Exercise: Solution (3)

In order to prove that **at least one of the two trolls tells the truth**, we show that the theory in which both are lying has **no model**

Let consider the negations of the troll's statements

$$\neg(\neg K \vee S) \equiv (K \wedge \neg S)$$

$$\neg(KA \vee KB) \equiv (\neg KA \wedge \neg KB)$$

Consider the theory  $\mathcal{A}'$  obtained from the above statements:

$$\mathcal{A}' = \{(K \wedge \neg S), (\neg KA \wedge \neg KB), \neg(KA \vee KB) \vee K, \neg K \vee (KA \vee KB)\}$$

**Claim:** The theory  $\mathcal{A}'$  has no model

- By contradiction, suppose such model exists. Call it  $I$
- Since  $I((K \wedge \neg S)) = \text{true}$  we have  $I(K) = \text{true}$
- Since  $I((\neg KA \wedge \neg KB)) = \text{true}$  then  $I((KA \vee KB)) = \text{false}$ ,
- Since  $I((KA \vee KB)) = \text{false}$  and  $I(\neg K \vee (KA \vee KB)) = \text{true}$  then  $I(\neg K) = \text{true}$  contradicting the fact that  $I$  is a (propositional) interpretation.

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## First-Order Logic

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. **Introduction to First-order Logic**
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. **Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# Propositional Logic (Recap)

## Exercise 0 (our friends Ander and Bjorn)

On a walk in the woods, you stumble upon a bridge guarded by two trolls. They say:

**Ander:** If I wear a tie or cufflinks then I wear a jacket.

**Bjorn:** Ander is not wearing a jacket!

Prove or disprove that:

1. Ander wears cufflinks.
2. Ander wears a tie.

## Exercise 0: Solution

On a walk in the woods, you stumble upon a bridge guarded by two trolls. They say:

**Ander:** If I wear a tie or cufflinks then I wear a jacket.

**Bjorn:** Ander is not wearing a jacket!

- Let  $C$  = Cufflink,  $T$  = Tie,  $J$  = Jacket
- Consider the theory  $\{\neg(C \vee T) \vee J, \neg J\}$
- In all the models  $I$  of the theory above we have that  $I(J) = \text{false}$
- Therefore,  $I(\neg(C \vee T)) = \text{true}$ .
- **By De Morgan:**  $\neg(C \vee T) \equiv (\neg C \wedge \neg T)$

# Predicate Logics: An Example

# The Need For Predicates

With propositional logic, we can formally reason about the truth or falsity of a sentence starting from the truth or falsity of some basic facts

- Basic facts are represented by propositional variables
- Logical connectives are used to represent how the facts are connected

Often (especially in databases) we need to talk about **properties of sets**

- Instead of propositional variables we use **constants** and **predicates**, i.e., *syntactic representations* of **individuals** and of **sets of individuals**, respectively.

## Example of Predicates

Suppose you want to formalize the fact that Fabio is a person that lives in a city called Bergamo (the subscript indicates the *arity* of the predicate).

- **Predicates:** Person<sub>/1</sub>, City<sub>/1</sub>, Lives<sub>/2</sub>.
- **Constants:** Fabio, Bergamo

Person(Fabio)  $\wedge$  City(Bergamo)  $\wedge$  Lives(Fabio, Bergamo)  
– This formula could capture our intuition.

## Example of Predicates

Clearly one could write different sentences using the same predicates.

- **Predicates:** Person<sub>/1</sub>, City<sub>/1</sub>, Lives<sub>/2</sub>.

Person(Bergamo)

- Bergamo is a Person.

City(Pavia)  $\wedge$  Lives(Fabio, Pavia)

- Fabio lives in Pavia.

## Example of Functions

We may also want to talk about objects we do not explicitly know. In this case we use functions.

- **Predicates:** Person<sub>/1</sub>, City<sub>/1</sub>, Lives<sub>/2</sub>.
- **Constants:** Fabio, Bergamo
- **Functions:** FatherOf<sub>/1</sub>,

Person(FatherOf(Fabio))

- The father of Fabio is a person.

City(Pavia)  $\wedge$  Lives(FatherOf(Fabio), Pavia)

- The father of Fabio lives in Pavia.

## Truth or Falsity

Consider the following formula

- Person(Fabio)  $\wedge$  City(Bergamo)  $\wedge$  Lives(Fabio, Bergamo)

**How can we define the truth value of such a formula?**

Instead of evaluating each atom, we evaluate the value of the predicates on the objects represented by the constants and functions.

# First-Order Logic

There are different formalisms to represent predicates.

First-order logic (FOL) is the logic to speak about objects.

FOL is concerned with properties of these objects and relations among them.

FOL uses functions (including constants) that denote objects.

# First-Order Logic: Syntax

# Syntax of First-Order Logic (1)

First-Order Logic formulae consists of the following components

1. **Terms**, intuitively representing individuals
2. **Predicates**, intuitively representing relations and properties of individuals
3. **Connectives**, intuitively representing the structure of the formula.
4. **Quantifiers**, intuitively representing individuals not explicitly occurring in the formula

## Syntax of First-Order Logic (2)

In what follows we assume to have the following pairwise distinct countable sets:

- the set of symbols **Vars** =  $\{x_1, x_2 \dots\}$
- the set of function symbols **Funcs**
  - A function symbol  $f$  has an associated *arity*  $\text{ar}(f)$  ( $f_{/\text{ar}(f)}$ )
  - Functions of arity 0 are called **constants**.
- the set of predicate symbols **Pred**
  - A predicate symbol  $P$  has an associated *arity*  $\text{ar}(P)$  ( $P_{/\text{ar}(P)}$ )

Note: a set is called *countable* if it is finite or countably infinite

# Terms

**Def:** The set **Terms** is inductively defined as follows.

- **Vars**  $\subseteq$  **Terms**
  - If  $t_1, \dots, t_n \in \mathbf{Terms}$  and  $f_{/n} \in \mathbf{Funcs}$  then  $f(t_1, \dots, t_n) \in \mathbf{Terms}$ .
  - Nothing else is in **Terms**.
- 

Further on this course, we will consider the fragment of FOL that includes only **constants** as functions (a **constant** is a function with arity 0).

In such case, we will simplify our formalization saying that the set **Terms** corresponds to **Cons**  $\cup$  **Vars** where **Cons** is a countable set of constants symbols

## Exercise

Assume **Vars** =  $\{x_1, x_2\}$

Assume **Funcs** =  $\{+_{/2}, *_{/2}, 1_{/0}\}$

Which of the following is in **Terms**?

- $+ (1, x_1)$
- $* (+ (1, 1), 2)$
- $+ (+ (x_1, x_1), + (x_2, x_2))$
- $* (+ (x_1, 1), + (1))$
- $* (+ (x_1, x_2), + (x_3, x_4))$

## Exercise: Solution

Assume **Vars** =  $\{x_1, x_2\}$

Assume **Funcs** =  $\{+_{/2}, *_{/2}, 1_{/0}\}$

Which of the following is in **Terms**?

- $+ (1, x_1)$ . **YES!**
- $* (+ (1, 1), \textcolor{red}{2})$ . **NO!**
- $+ (+ (x_1, x_1), + (x_2, x_2))$ . **YES!**
- $* (+ (x_1, 1), \textcolor{red}{+ (1)})$ . **NO!**
- $* (+ (x_1, x_2), + (\textcolor{red}{x_3}, \textcolor{red}{x_4}))$ . **NO!**

# FOL Formulae

The set of **Forms** is defined inductively as follows:

If  $t_1, \dots, t_n \in \text{Terms}$  and  $P_{/n} \in \text{Pred}$ , then  $P(t_1, \dots, t_n) \in \text{Forms}$

If  $t_1, t_2 \in \text{Terms}$  then  $(t_1 = t_2) \in \text{Forms}$

If  $f_1, f_2 \in \text{Forms}$  and  $x \in \text{Vars}$  then

- $(f_1 \wedge f_2) \in \text{Forms}$
- $(f_1 \vee f_2) \in \text{Forms}$
- $(f_1 \rightarrow f_2) \in \text{Forms}$
- $\neg(f_1) \in \text{Forms}$
- $\exists x. f_1 \in \text{Forms}$
- $\forall x. f_1 \in \text{Forms}$ .

Nothing else is in **Forms**

} These are called  
**atomic formulae**

# Functions and Predicates

The sets **Preds**, **Funcs**, and **Vars** form the **alphabet** of our logic.

At the syntactic level predicates and functions look similar

- There is however a difference in the syntactic rules that define them

The difference will be clear at the **semantic level**

- Functions define individuals
- Predicates define relations among individuals

# Propositional and Predicate Logics

- Connectives work the same in both logics
  - At the syntactic level
- A predicate  $P$  with  $\text{ar}(0)$  can be seen as propositional variables.
  - First-Order Logic is an extension of Propositional Logic
- In addition we can talk about objects:
  - Constants and variables representing specific objects
    - $\exists x$  **existential quantification**: there exists an object such that.
    - $\forall x$  **universal quantification**: all objects such that.

# Exercise

Assume the following alphabet:

$$\mathbf{Vars} = \{x, y\}, \mathbf{Funcs} = \{+_{/2}, *_{/2}, 1_{/0}, 2_{/0}, \dots\}, \mathbf{Preds} = \{\text{Even}_{/1}, \text{Odd}_{/1}\}$$

Which of the following are FOL formulae?

1.  $\text{Odd}(2)$
2.  $\neg \text{Odd}(2) \vee \neg \text{Nat}(1)$
3.  $\neg \vee \text{Odd}(3)$
4.  $\forall x. \exists y. \text{Odd}(+(x, y)) \wedge \text{Nat}(x) \wedge \text{Nat}(y)$

## Exercise: Solution

Assume the following alphabet:

$$\mathbf{Vars} = \{x, y\}, \mathbf{Funcs} = \{+_{/2}, *_{/2}, 1_{/0}, 2_{/0}, \dots\}, \mathbf{Preds} = \{\text{Even}_{/1}, \text{Odd}_{/1}\}$$

Which of the following are FOL formulae?

1.  $\text{Odd}(2)$  **YES!**
2.  $\neg \text{Odd}(2) \vee \neg \text{Nat}(1)$  **YES!**
3.  $\neg \vee \text{Odd}(3)$  **NO!**
4.  $\forall x. \exists y. \text{Odd}(+(x, y)) \wedge \text{Nat}(x) \wedge \text{Nat}(y)$  **YES!**

## Exercise: observation

Which of the following are FOL formulae?

1.  $\text{Odd}(2)$  **YES!**
2.  $\neg \text{Odd}(2) \vee \neg \text{Nat}(1)$  **YES!**
3.  $\neg \vee \text{Odd}(3)$  **NO!**
4.  $\forall x. \exists y. \text{Odd}((x, y)) \wedge \text{Nat}(x) \wedge \text{Nat}(y)$  **YES!**

**Can we establish whether the above formulae are true?**

## Exercise: observation

Which of the following are FOL formulae?

1.  $\text{Odd}(2)$  **YES!**
2.  $\neg \text{Odd}(2) \vee \neg \text{Nat}(1)$  **YES!**
3.  $\neg \vee \text{Odd}(3)$  **NO!**
4.  $\forall x. \exists y. \text{Odd}((x, y)) \wedge \text{Nat}(x) \wedge \text{Nat}(y)$  **YES!**

**Can we establish whether the above formulae are true?**

**Not yet!** For now, formulae are meaningless syntactic objects.

To give meaning to the symbols we need **interpretations** that, similarly to Propositional logic, will tell us whether a formula is **true** or **false**.

# **First-Order Logic: Semantics**

# Domains and Interpretations

In order to interpret FOL formulae, we need a set of objects.

We will call this set **domain of discourse** (or simply domain).

- The domain of natural numbers ...
- The domain of people and their jobs ...

**Observe:** **objects** (i.e., elements in the domain) and **terms** (elements of the set **Terms** define earlier) are not the same!

- Function  $2_{/0}$  is not the number 2 in the domain of natural numbers.

To connect **terms** with **objects** we use **interpretations**.

- Intuitively, interpretations connect **terms** to their “**meaning**”

# Relations

Before providing the notion of interpretation, we need the following notion of **relation**

Assume to have a set  $\mathcal{A}$  of elements

An **n-ary tuple** over  $\mathcal{A}$  is a sequence of **n** elements of  $\mathcal{A}$

- Example:  $(a_1, a_2, \dots, a_n)$

The **n-th power of  $\mathcal{A}$**  (written  $\mathcal{A}^n$ ) is the set of **n-ary tuples** over  $\mathcal{A}$

An **n-ary relation** over  $\mathcal{A}$  is a **subset** of  $\mathcal{A}^n$

# FOL Interpretation: Definition

Assume sets **Funcs** and **Preds** of functions and predicates, respectively

**Definition:** An **interpretation**  $I$  is a pair  $(\Delta^I, \cdot^I)$  where

- $\Delta^I$  is a countable set (the domain of discourse)
- $\cdot^I$  is a function from **Funcs**  $\cup$  **Preds** defined as follows:
  - $f^I = \Delta^k \rightarrow \Delta$ , for each  $f \in \text{Funcs}$  with  $ar(f) = k$
  - $P^I \subseteq \Delta^k$ , for each  $P \in \text{Preds}$  with  $ar(P) = k$

So, the function  $\cdot^I$  maps each function symbol  $f$  of arity  $n$  to an  $n$ -ary function  $f^I = \Delta^k \rightarrow \Delta$  and each predicate symbol  $P$  of arity  $n$  to an  $n$ -ary relation  $P^I \subseteq \Delta^k$ .

In case for a function  $f$  we have that  $ar(f) = 0$ , then  $f^I$  denotes exactly one object in  $\Delta^I$ . As already said, we call them **constants**.

# Example of Interpretation

Assume the following:

$$\mathbf{Funcs} = \{+, \cdot, 1_0, 2_0, \dots\}, \mathbf{Preds} = \{\text{Even}_1, \text{Odd}_1\}$$

In the standard interpretation  $I$  of natural numbers we have:

- $\Delta^I$  is equal to  $\mathbb{N}$
- The predicate  $\text{Even}_1$  is mapped to the unary relation  $\text{Even}^I \subseteq \Delta$ , representing the set of even numbers
- The predicate  $\text{Odd}_1$  is mapped to the unary relation  $\text{Odd}^I \subseteq \Delta$ , representing the set of odd numbers
- Each constant  $n_0$  is mapped to a natural number:  $n^I = n \in \mathbb{N}$

**What is  $I$  telling us about the following formulae?**

# Interpretation of Formulae

Can we say whether the following formulae are **true** in  $I$ ?

1.  $Odd(2)$
2.  $\neg Odd(2) \vee \neg Odd(1)$
3.  $\exists x. Odd(x) \vee Even(x)$
4.  $Odd(x) \vee Even(x)$

# Interpretation of Formulae

Can we say whether the following formulae are **true** in  $I$ ?

1.  $\text{Odd}(2)$
2.  $\neg \text{Odd}(2) \vee \neg \text{Odd}(1)$
3.  $\exists x. \text{Odd}(x) \vee \text{Even}(x)$
4.  $\text{Odd}(x) \vee \text{Even}(x)$

Intuitively,  $I$  tells everything we need to know about formulae 1 and 2. Moreover, by interpreting  $\exists x$  as “**there exists a domain element**”, we have that  $I$  still gives us the truth value of the formula 3.

Formula 4 is totally different: **we need a way to evaluate variables**.

# Assignments

Assume an interpretation  $I = (\Delta^I, \cdot^I)$  for **Funcs** and **Preds**.

An **assignment**  $\alpha$  is function that maps each variable symbol in **Vars** to an object in  $\Delta^I$ , i.e.,

$$\alpha: \mathbf{Vars} \rightarrow \Delta^I$$

We define the extension  $\hat{\alpha}$  of  $\alpha$  to **Terms** as follows.

- $\hat{\alpha}(x) = \alpha(x)$  , for each  $x \in \mathbf{Vars}$ .
- $\hat{\alpha}(f(t_1, \dots, t_n)) = f^I(\hat{\alpha}(t_1), \dots, \hat{\alpha}(t_n))$ , for each  $f \in \mathbf{Terms}$ .

Given a variable  $x \in \mathbf{Vars}$  and an object  $o \in \Delta^I$ , we define the assignment  $\alpha[x \rightarrow o]$  as

- $\alpha[x \rightarrow o](y) = \alpha(y)$ , for each variable  $y \neq x$
- $\alpha[x \rightarrow o](x) = o$

# Truth of a Formula

Assume a FOL formula  $\varphi$ , an assignment  $\alpha$ , and an interpretation  $I$

We say that  $\varphi$  is true in  $I$  according to  $\alpha$  (written  $I, \alpha \models \varphi$ ) if the following holds:

- $I, \alpha \models P(x_1, \dots, x_n)$  and  $(\hat{\alpha}(x_1), \dots, \hat{\alpha}(x_n)) \in P^I$
- $I, \alpha \models (x_1 = x_2)$  and  $\hat{\alpha}(x_1) = \hat{\alpha}(x_2)$
- $I, \alpha \models (\varphi_1 \wedge \varphi_2)$  and  $I, \alpha \models \varphi_1$  and  $I, \alpha \models \varphi_2$
- $I, \alpha \models (\varphi_1 \vee \varphi_2)$  and  $I, \alpha \models \varphi_1$  or  $I, \alpha \models \varphi_2$
- $I, \alpha \models (\varphi_1 \rightarrow \varphi_2)$  and either  $I, \alpha \not\models \varphi_1$  or  $I, \alpha \models \varphi_2$
- $I, \alpha \models \neg\varphi_1$  and  $I, \alpha \not\models \varphi_1$  ( $\varphi$  is NOT true in  $I$  according to  $\alpha$ )
- $I, \alpha \models \exists x. \varphi_1(x)$  and  $I, \alpha[x \rightarrow a] \models \varphi_1$ , for some  $a \in \Delta^I$
- $I, \alpha \models \forall x. \varphi_1(x)$  and  $I, \alpha[x \rightarrow a] \models \varphi_1$ , for each  $a \in \Delta^I$

# Truth and Falsity

Assume a FOL formula  $\varphi$ .

- $\varphi$  is called **satisfiable** if  $I, \alpha \models \varphi$ , for some  $I, \alpha$ .
- $\varphi$  is called **unsatisfiable** if  $I, \alpha \models \neg\varphi$ , for every  $I, \alpha$ .
- $\varphi$  is called **valid (tautology)** if  $I, \alpha \models \varphi$ , for every  $I, \alpha$ .
- $\varphi$  is called **falsifiable** if  $I, \alpha \models \neg\varphi$ , for some  $I, \alpha$ .

# Equalities – Symbols and Semantics

Given an interpretation  $I$  and an assignment  $\alpha$ , we know that

$$I, \alpha \vDash (t_1 = t_2) \text{ if } \hat{\alpha}(t_1) = \hat{\alpha}(t_2)$$

What is the difference between the two equality symbols above?

- The  $=$  symbol on the left is **syntactic**, indeed, it represents a binary relation symbol with special interpretation.
- The  $=$  symbol on the right is **semantic**, it means **identity** over the domain of  $I$

## Exercise

Assume functions  $t_{/0}$  and  $s_{/0}$

Are the following formulae satisfiable, falsifiable, unsatisfiable, or valid?

1.  $(t = t)$
2.  $(t = s)$
3.  $\neg(t = t)$
4.  $\neg(t = s)$

Give examples of interpretations to support your claims

## Exercise

Assume functions  $t_{/0}$  and  $s_{/0}$

Are the following formulae satisfiable, falsifiable, unsatisfiable, or valid?

1.  $(t = t)$       **Valid (is a tautology)**
2.  $(t = s)$       **Satisfiable, Falsifiable**
3.  $\neg(t = t)$       **Unsatisfiable**
4.  $\neg(t = s)$       **Satisfiable, Falsifiable**

Give examples of interpretations to support your claims

## Truth Of a Formula: Additional Remarks

- Connectives work as in Propositional Logic.
  - Once we evaluate atoms, we essentially have the truth value of a propositional formula.
- To capture the intuitive meaning of quantification we use assignments.

## **Example of First-Order Formulae**

## Alphabet and Domain

- **Funcs** =  $\{Arethi_{/0}, Bob_{/0}, Dep1_{/0}, Dep2_{/0}\}$
- **Preds** =  $\{Employee_{/1}, Department_{/1}, Works_{/2}, Directs_{/2}\}$
- $\Delta^I = \{a, b, c, d1, d2\}$
- **Observe:** None of the constants are in  $\Delta^I$ .

## Some Questions

- Is *Dep1* a department or an employee?
- Can a constant be an object?

## Some Questions

- Is *Dep1* a department or an employee?
  - Not necessarily. Depends on the interpretations.
- Can a constant be an object?
  - *Only if we assume to have the same object in the domain!*

## Interpretation: Functions

- **Funcs** = {Arethi<sub>/0</sub>, Bob<sub>/0</sub>, Dep1<sub>/0</sub>, Dep2<sub>/0</sub>}
  - $\Delta^I = \{a, b, c, d1, d2\}$
  - Arethi<sup>I</sup> =  $a$
  - Bob<sup>I</sup> =  $b$
  - Dep1<sup>I</sup> =  $d1$
  - Dep2<sup>I</sup> =  $d2$
- **Observe:** we have no function for  $c$ !

## Interpretation: Predicates

**Preds** = { Employee<sub>/1</sub>, Department<sub>/1</sub>, Works<sub>/2</sub>, Directs<sub>/2</sub> }

- $\Delta^I = \{a, b, c, d1, d2\}$
- $Employee^I = \{a, b, c\}$
- $Department^I = \{d1, d2\}$
- $Works^I = \{(a, d1), (b, d2), (c, d1)\}$
- $Directs^I = \{(a, d1), (c, d1), (c, d2)\}$

**Observe:**  $c$  takes part to the interpretation of predicates!

## Exercise

Assume an assignment  $\alpha(x) = a$ .

Which of the following formulae are true in  $I, \alpha$  ?

1.  $(x = Arethi) \vee (x = Bob)$
2.  $Employee(Arethi) \wedge Works(Arethi, Dep1)$
3.  $\exists x. Directs(x, Dep2)$
4.  $\forall x. Employee(x) \rightarrow \neg Department(x)$
5.  $\forall x. \forall y. \forall z. Directs(x, y) \wedge Directs(x, z) \rightarrow y = z$

## Exercise: Solution

Assume the assignment  $\alpha(x) = a$ .

- $(x = Arethi) \vee (x = Bob)$
- $\hat{\alpha}(Arethi) = a$
- $\hat{\alpha}(Bob) = b$

**We can conclude that the formula is true.**

Observe, this depends on the assignment → Consider for instance a different assignment  $\beta(x) = d1$

## Exercise: Solution

Assume the assignment  $\alpha(x) = a$ .

- $Employee(Arethi) \wedge Works(Arethi, Dep1)$
- $\hat{\alpha}(Arethi) = a$
- $\hat{\alpha}(Dep1) = d1$
- $a \in Employee^I$  **and**  $(a, d1) \in Works^I$

We can conclude that the formula is true.

Observe, this holds for every assignment (we have no variables)!

## Exercise: Solution

Assume the assignment  $\alpha(x) = a$ .

- $\exists x. Directs(x, Dep2)$  is true if for some  $o \in \Delta^I$  we have

$$I, \alpha[x = o] \models Directs(x, Dep2)$$

- Since  $I, \alpha[x = c] \models Directs(x, Dep2)$ , since we have that  $Dep2^I = d2$  and  $(c, d2) \in Directs^I$  **we can conclude that the formula is true.**

**Observe:** the original assignment for  $x$  does not really matter.

## Exercise: Solution

Assume the assignment  $\alpha(x) = a$ .

- $\forall x. Employee(x) \rightarrow \neg Department(x)$  is true if for every  $o \in \Delta^I$  we have:

$$I, \alpha[x = o] \models Employee(x) \rightarrow \neg Department(x)$$

- This is the case, therefore the formula is **true**.

Intuitively, the formula requires that the two sets are **disjoint**.

## Exercise: Solution

Assume the assignment  $\alpha(x) = a$ .

- $\forall x. \forall y. \forall z. \text{Directs}(x, y) \wedge \text{Directs}(x, z) \rightarrow y = z$  is true if for every  $o, o', o'' \in \Delta^I$  we have:  
 $I, \alpha[x = o][y = o'][z = o''] \models \text{Directs}(x, y) \wedge \text{Directs}(x, z) \rightarrow y = z$
- This is not the case for  $\alpha[x = c][y = d1][z = d2]$ . So, the formula is **false!**

Intuitively, the formula is a **key constraint** (see Database course)

# **Consequence and Equivalence**

# Implication and Equivalence

Let  $\varphi, \psi$  be two FOL formulae. We have that:

- $\varphi$  **implies**  $\psi$  (written  $\varphi \vDash \psi$ ) if for every  $I, \alpha$  s.t.  $I, \alpha \vDash \varphi$  we have  $I, \alpha \vDash \psi$
- $\varphi$  **is equivalent to**  $\psi$  ( $\varphi \equiv \psi$ ) if for every  $I, \alpha$ , we have that  $I, \alpha \vDash \varphi$  if and only if  $I, \alpha \vDash \psi$

We can extend **implication** and **equivalence** to **logical theories** (finite sets of logical formulae) in the natural way.

# Useful Equivalences

Some useful equivalences for FOL Formulae

- **De Morgan’s Law 1:**  $\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi)$
- **De Morgan’s Law 2:**  $\neg(\varphi \vee \psi) \equiv (\neg\varphi \wedge \neg\psi)$
- **Double Negation:**  $\neg\neg\varphi \equiv \varphi$
- **Negation of Existential:**  $\neg\exists x. \varphi \equiv \forall x. \neg\varphi$
- **Negation of Universal:**  $\neg\forall x. \varphi \equiv \exists x. \neg\varphi$

# Exercise

Prove De Morgan's Law 1 and the law of double negation.

# Exercise: Solution

Prove De Morgan's Law 1 and the law of double negation.

De Morgan's Law 1:  $\neg(\varphi \wedge \psi) \equiv (\neg\varphi \vee \neg\psi)$

**Proof:** We prove the claim showing the truth tables.

	Truth table $\neg(\varphi \wedge \psi)$	Truth table $(\neg\varphi \vee \neg\psi)$
• $\varphi = \text{true}, \psi = \text{true}$ .	false	false
• $\varphi = \text{true}, \psi = \text{false}$	true	true
• $\varphi = \text{false}, \psi = \text{true}$	true	true
• $\varphi = \text{false}, \psi = \text{false}$	true	true

# Exercise: Solution

Prove De Morgan's Law 1 and the law of double negation.

Double Negation:  $\neg\neg\varphi \equiv \varphi$

**Proof:** We prove the claim showing the truth tables.

- $\varphi = \text{false}$
- $\varphi = \text{true}$

Truth table  $\varphi$

false
true

Truth table  $\neg\neg\varphi$

false
true

# Exercise

Prove Negation of Existential

$$\neg \exists x. \varphi \equiv \forall x. \neg \varphi$$

# Exercise

Prove Negation of Existential

$$\neg \exists x. \varphi \equiv \forall x. \neg \varphi$$

**Proof:** If  $I, \alpha \models \neg \exists x. \varphi$  then  $I, \alpha \not\models \exists x. \varphi$ . Therefore,  $I, \alpha[x = a] \models \neg \varphi$ , for every  $a \in \Delta^I$ . In turn, this proves that  $I, \alpha \models \forall x. \neg \varphi$ .

If  $I, \alpha \models \forall x. \neg \varphi$ , then  $I, \alpha[x = a] \models \neg \varphi$ , for every  $a \in \Delta^I$ . In turn, this proves that  $I, \alpha \not\models \exists x. \varphi$ .

# **Introduction to computational complexity**

# Computational complexity (1/2)

**Computational complexity theory** aims to study how difficult it is to solve specific problems.

Complexity theory deals with **decision problems**: i.e., problems that admit a **yes/no** answer.

A **decision algorithm** is an algorithm that computes the correct truth value for each input instance of a **decision problem** (**The algorithm has to terminate on all inputs**):

- **input**: an instance of the problem
- **output**: **yes** or **no**

A decision problem is **decidable** if there exists a decision algorithm for it. Otherwise it is **undecidable**.

The complexity is measured in terms of the amount of **resources** (time, space) that the algorithm needs to solve the problem (complexity of the algorithm, or **upper bound**).

To measure the complexity of the problem, we consider the **best possible algorithm** that solves it (**lower bound**).

## Computational complexity (2/2)

**Worst-case complexity analysis:** the complexity is measured in terms of a (complexity) function  $f$ :

- **argument:** the size  $n$  of an instance of the problem
- **result:** the amount  $f(n)$  of time/space needed in the worst-case to solve an instance of size  $n$

To abstract away from contingent issues (e.g., programming language, processor speed, etc.), we refer to an abstract computing model: **Turing Machines** (TMs).

Usually one does not consider specific complexity functions  $f$ , but rather families **C** of complexity functions, giving rise to complexity classes.

**Definition:** A **time/space complexity class C** is the set of all problems  $P$  such that an instance of  $P$  of size  $n$  can be solved in time/space at most  $C(n)$ .

# Reductions

To establish **lower bounds** on the complexity of problems, we make use of the notion of reduction:

- **Definition:** A **reduction** from a problem P1 to a problem P2 is a function R from instances of P1 to instances of P2 such that:
  - R is efficiently computable (typically in logarithmic space), and
  - An instance I of P1 has answer yes if and only if R(I) has answer yes.

We say that P1 **reduces to** P2 if there is a reduction R from P1 to P2.

- **Intuition:** If P1 reduces to P2, then P2 is at least as difficult as P1, since we can solve an instance I of P1 by reducing it to the instance R(I) of P2 and then solve R(I).

# Hardness and Completeness

If we can provide an algorithm that solve a problem P of size n by using at most  $C(n)$  time\space, than we can prove the **membership** of P to the class C (the **upper-bound**)

To provide a **lower-bound** we need to refer to the notion of **hardness**:

- **Definition:** A problem P is **hard** for a complexity class C if every problem in C can be reduced to P.

If we have both, we show the **completeness** w.r.t. a complexity class

- **Definition:** A problem P is **complete** for a complexity class C if it is **hard** for C, and it **belongs** to C (membership to C)

Intuitively, a problem that is complete for C is among the hardest problems in C.

# Tractability and intractability: PTime and NP

**Definition:** **PTime** is the set of problems solvable in polynomial time by a **deterministic TM**.

- These problems are considered **tractable**, i.e., solvable for large inputs.

**Definition:** **NP** is the set of problems solvable in polynomial time by a **non-deterministic TM**.

- These problems are believed **intractable**, i.e., unsolvable for large inputs.
- The best known actual algorithms actually require exponential time.
- Corresponds to a large class of practical problems, for which the following type of algorithm can be used:
  1. Non-deterministically guess a possible solution of polynomial size.
  2. Check in polynomial time that the guessed solutions is good.

## Complement of problems in NP: coNP

**Definition:** coNP is the set of problems whose complement is in NP, i.e., problems for which determining whether an instance admits a **no** answer is in NP.

For problems whose complexity is characterized in terms of a non-deterministic TM, solving the problem and solving its complement might be different.

The reason for this is that a **yes** answer is returned if there exists a non-deterministic computation-path of the TM that leads to acceptance. Instead, a **no** answer requires that all non-deterministic computation-paths of the TM lead to rejection.

Specifically, coNP is believed to be different from both NP and PTime.

# Complexity classes above NP

**Definition:** **PSpace** is the set of problems solvable in polynomial space by a deterministic TM.

- Polynomial space is "**not really good**", since these problems may require exponential time. Indeed, these problems are believed to be more difficult than NP problems.

**Definition:** **ExpTime** is the set of problems solvable in exponential time by a deterministic TM.

- These problems are considered to be very difficult.

**Definition:** **NExpTime** is the set of problems solvable in exponential time by a non-deterministic TM.

# Complexity classes below PTime

**Definition:** **LogSpace** (**NLogSpace**) is the set of problems solvable in logarithmic space by a (non-)deterministic TM.

- **Note:** when measuring the space complexity, the size of the input does not count, and only the working memory (TM tape) is considered.

**Definition:** **AC<sup>0</sup>** is the set of problems solvable in constant time using a polynomial number of processors.

- These problems are solvable efficiently even for very large inputs.
- Corresponds to the complexity of model checking a **fixed FO formula** when the input is the model only.

# Relationship between the complexity classes

The following relationships are known:

$$\begin{aligned} \text{AC}^0 &\subsetneq \text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{PTIME} \subseteq \\ &\subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \\ &\subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \end{aligned}$$

Moreover, we know that:

$$\text{PTIME} \subsetneq \text{EXPTIME}.$$

# Complexity of First-Order Logic

# Logical Formulae and Logical Theories

In mathematics, logic is used mostly to describe a set of “relevant” interpretations and prove theorems on these interpretations.

We define a **logical theory**, i.e., a finite set of logical formulae.

We look at interpretations satisfying the theory

- independently from the chosen assignment
- Interpretations satisfying a theory represent **valid possible “worlds”**.

We use the formal tools of equivalence and implication to prove our statements.

# Implication and Equivalence (recap)

Let  $\varphi, \psi$  be two FOL formulae,

- $\varphi$  **implies**  $\psi$  ( $\varphi \vDash \psi$ ) if for every  $I, \alpha$  s.t.  $I, \alpha \vDash \varphi$  we have  $I, \alpha \vDash \psi$
- $\varphi$  **is equivalent to**  $\psi$  ( $\varphi \equiv \psi$ ) if for every  $I, \alpha$ , we have that  $I, \alpha \vDash \varphi$  if and only if  $I, \alpha \vDash \psi$

# An Example of FOL Theory

1.  $\forall x, y, z ((x + y) + z = x + (y + z))$ , i.e., addition is associative.
2.  $\forall x, y (x + y = y + x)$ , i.e., addition is commutative.
3.  $\forall x, y, z ((x \cdot y) \cdot z = x \cdot (y \cdot z))$ , i.e., multiplication is associative.
4.  $\forall x, y (x \cdot y = y \cdot x)$ , i.e., multiplication is commutative.
5.  $\forall x, y, z (x \cdot (y + z) = (x \cdot y) + (x \cdot z))$ , i.e., multiplication distributes over addition.
6.  $\forall x (x + 0 = x \wedge x \cdot 0 = 0)$ , i.e., zero is an identity for addition, and an absorbing element for multiplication (actually superfluous<sup>[note 1]</sup>).
7.  $\forall x (x \cdot 1 = x)$ , i.e., one is an identity for multiplication.
8.  $\forall x, y, z (x < y \wedge y < z \Rightarrow x < z)$ , i.e., the ' $<$ ' operator is transitive.
9.  $\forall x (\neg(x < x))$ , i.e., the ' $<$ ' operator is irreflexive.
10.  $\forall x, y (x < y \vee x = y \vee y < x)$ , i.e., the ordering satisfies trichotomy.
11.  $\forall x, y, z (x < y \Rightarrow x + z < y + z)$ , i.e. the ordering is preserved under addition of the same element.
12.  $\forall x, y, z (0 < z \wedge x < y \Rightarrow x \cdot z < y \cdot z)$ , i.e. the ordering is preserved under multiplication by the same positive element.
13.  $\forall x, y (x < y \Rightarrow \exists z (x + z = y))$ , i.e. given any two distinct elements, the larger is the smaller plus another element.
14.  $0 < 1 \wedge \forall x (x > 0 \Rightarrow x \geq 1)$ , i.e. zero and one are distinct and there is no element between them.
15.  $\forall x (x \geq 0)$ , i.e. zero is the minimum element.

# Logical Tasks: Complexity

- **Validity:** check whether a FOL formula is valid.
- **Satisfiability:** check whether a FOL formula is satisfiable.
- **Implication:** Check whether  $\varphi \models \psi$ , for input FOL formulae  $\varphi, \psi$ .

# Logical Tasks: Complexity

- **Validity:** check whether a FOL formula is valid.
  - Undecidable
- **Satisfiability:** check whether a FOL formula is satisfiable.
  - Undecidable
- **Implication:** Check whether  $\varphi \models \psi$ , for input FOL formulae  $\varphi, \psi$ .
  - Undecidable

Unfortunately, in general none of the problems above is decidable.

# Complexity of Propositional Logic

# Truth and Falsity

Assume a propositional formula  $f$ .

- $f$  is called **satisfiable** if  $I(f) = \text{true}$ , for some propositional interpretation  $I$ .
- $f$  is called **valid (tautology)** if  $I(f) = \text{true}$ , for every propositional interpretation  $I$ .
- $f$  is called **falsifiable** if  $I(f) = \text{false}$ , for some propositional interpretation  $I$ .
- $f$  is called **unsatisfiable** if  $I(f) = \text{false}$ , for every propositional interpretation  $I$ .

# Computational Complexity: Upper Bounds

- Checking whether a propositional formula  $f$  is **satisfiable** is in NP.
  - Guess an interpretation  $I$  (for the variables in  $f$ ).
  - Check whether the  $I(f) = \text{true}$ .
- Checking whether a propositional formula  $f$  is **unsatisfiable** is in coNP.
  - Guess an interpretation  $I$  (for the variables in  $f$ ).
  - Check whether the  $I(f) = \text{true}$ .
- Checking whether a propositional formula  $f$  is **falsifiable** is in NP.
  - Guess an interpretation  $I$  (for the variables in  $f$ ).
  - Check whether the  $I(f) = \text{false}$ .
- Checking whether a propositional formula  $f$  is **valid** is in coNP.
  - Guess an interpretation  $I$  (for the variables in  $f$ ).
  - Check whether the  $I(f) = \text{false}$ .

# Normal Forms

- A propositional formula  $f$  is in  $n$  conjunctive normal form ( $n$ -CNF) if
  - $f = (I_1^1 \vee I_2^1 \vee \dots \vee I_n^1) \wedge \dots \wedge (I_1^m \vee I_2^m \vee \dots \vee I_n^m)$
  - Where each  $I_j^i$  is either an atom or its negation.
  - Every propositional formula has an equivalent formula in 3-CNF
  - However, the equivalent may be exponentially larger.
- A propositional formula  $f$  is in  $n$  disjunctive normal form ( $n$ -DNF) if
  - $f = (I_1^1 \wedge I_2^1 \wedge \dots \wedge I_n^1) \vee \dots \vee (I_1^m \wedge I_2^m \wedge \dots \wedge I_n^m)$
  - Where each  $I_j^i$  is either an atom or its negation.
  - Every propositional formula has an equivalent formula in 3-DNF
  - However, the equivalent may be exponentially larger.

# Computational Complexity: Lower Bounds

- Checking whether a propositional formula is **satisfiable** is NP-Hard.
  - Even for formulae in 3-CNF
- Checking whether a propositional formula is **unsatisfiable** is coNP-Hard.
  - Even for formulae in 3-DNF
- Checking whether a propositional formula is **falsifiable** is NP-hard.
  - Even for formulae in 3-DNF.
- Checking whether a propositional formula is **valid** is in coNP-Hard.
  - Even for formulae in 3-CNF

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## First-Order Logic and Databases (Relational Calculus)

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
- 3. Relational Calculus**
- 4. Information Integration Systems (IIS)**
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
- 9. Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# The Relational Model of Data

# Database Models

- The goal of Information Integration is to provide a uniform view of data coming from multiple and heterogenous sources.
  - SQL Technologies, Spreadsheets, XML\JSON, CSV, ...
- To abstract from the details of each single source containing the information, we often assume that data are contained in a **database**.
  - We can safely assume that the wrappers present the data this way.
- **What is a database?** To develop a formal theory of information integration, we need to formalize the notion of database.

# The Relational Model of Data

Proposed by E. F. Codd in 1970 for the **physical independence** of data. Indeed, the way data is used at the logical level is independent from their physical storage.

Made available in commercial **DBMSs** in 1981 (it is not easy to implement data independence efficiently and reliably!)

It is based on (a variant of) the mathematical notion of **relation**

Relations admit a very natural and intuitive representation as **tables**.

# Mathematical Relations

- Consider  $n$  sets  $D_1, D_2, \dots, D_n$ , not necessarily distinct.
- The **cartesian product**  $D_1 \times D_2 \times \dots \times D_n$ , is the set of **all**  $n$ -tuples  $(d_1, d_2, \dots, d_n)$  such that  $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n$ .
- A (mathematical) **relation** over  $D_1, D_2, \dots, D_n$  is a **subset of their cartesian product**  $D_1 \times D_2 \times \dots \times D_n$ .
- When  $\mathbf{R}$  is a mathematical relation over the sets  $D_1, D_2, \dots, D_n$ , such sets are called the **domains** or the relation  $\mathbf{R}$ . A relation over  $n$  domains has **arity**  $n$ .
- The number of  $n$ -tuples is the **cardinality** of the relation.

# Mathematical Relations: Example

- $D_1 = \{\text{Pizza, Ramen}\}$  (delicious food)
- $D_2 = \{\text{Italy, Japan}\}$  (countries)

Cartesian product  $D_1 \times D_2$  

Pizza	Italy
Pazza	Japan
Ramen	Italy
Ramen	Japan

A relation:  $\text{Origin} \subseteq D_1 \times D_2$  

Pizza	Italy
Ramen	Japan

# Relational Databases

In what follows we assume to have the following pairwise distinct countably infinite sets:

- **Cons** of constants
- **Rels** of relational predicates
- **Vars** of variables

and the set:

- **Terms = Vars  $\cup$  Cons**

# Relational Databases: Schema

## Relation schema:

- A relational predicate  $R \in \text{Rels}$  with a set of attributes  $X = \{A_1, \dots, A_n\}$  over the domains  $D_1, \dots, D_n$ .
- It is denoted by  $R(A_1:D_1, \dots, A_n:D_n)$  or simply  $R(A_1, \dots, A_n)$ .
- The integer  $n$  is called the **arity** of  $R$  and is denoted by  $\text{ar}(R)$ . We write  $R_n$  if  $R$  has  $\text{ar}(R) = n$ .

### Exams

Student	Grade	Course
---------	-------	--------



Exams(Student,Grade,Course)

## Relational Database schema:

a set of **relation schemas** all with different names  $S = \{R_1(X_1), \dots, R_n(X_n)\}$

# Relational Databases: Instances

## Relation:

- A **relation** over a schema  $R(X)$  such that  $ar(R) = n$  is a set  $r$  of  $n$ -tuples on  $X$ .
- It represents an **instance** of a relation schema

<u>Exams</u>	Student	Grade	Course
3456	30	04	
3254	24	05	

Instance of the relation Exams

## Database:

- A **database**  $D$  over a schema  $S = \{R_1(X_1), \dots, R_n(X_n)\}$  is a set of relations  $r = \{r_1, \dots, r_n\}$  (with  $r_i$  relation on  $R_i$ ). We denote with  $D(R_i)$  the relation  $r_i$ .
- Given a database  $D$  we denote by  $\text{dom}(D)$  the domain of  $D$ , i.e., the set of constants occurring in  $D$ . We always have that  $\text{dom}(D) \subseteq \text{Cons}$ .
- Sometimes in this course we will see a database as a set of facts of the form  $R(v_1, v_2, \dots, v_n)$  where  $(v_1, v_2, \dots, v_n)$  is a  $n$ -tuple over the relation schema  $R_n$

# Example of a database

**Products**

Code	Color
P001	Red
P002	Green
P003	Blue

**Customers**

Code	Name
C001	Eryn
C002	Dylan
C003	Cora

**Orders**

Code	Customer	Product	Quantity
O001	C001	P001	4
O002	C002	P002	2
O003	C001	P001	3

# Summary of Definitions

**Relation name:** a symbol representing the relation and its arity.

$$R_{/k}$$

**Database schema:** non-empty set of relation schemas with different names.

$$S = \{ R_1, \dots, R_n \}$$

**Tuple:** An ordered sequence of domain elements.

$$t = \langle v_1, \dots, v_n \rangle$$

**Relation (instance):** of  $R_{/k}$  is a set of n-tuples.

$$\{t_1, \dots, t_m\}$$

**Database (instance):** over  $S$ . A function  $D$  from relation names to instances.

$$D(R_1) = \dots, D(R_n) =$$

# Integrity Constraints

Not all database instances represent information that is consistent with our knowledge of the domain of interest

To define valid instances we can use **integrity constraints**, i.e., conditions written in a formal language that all valid database instances must respect

The relational model allows two different forms of constraints

- **Intra-relational**: condition specified within a single relation
- **Inter-relational**: condition specified across different relations

# Integrity Constraints: motivations

- Useful to describe the domain in **greater detail**
- A contribution to “**data quality**”
- An element in the **design process**
- Used by the system in choosing the strategy for **query processing**

**Note:** not all the properties can be described by means of integrity constraints

# Key Constraints

- **Intuition:** A **key constraint** is a statement that specifies that a (non-empty) set of tuple components must be unique in a relation.
- **Syntax:** Key:  $R[A_1, \dots, A_k]$  where  $A_1, \dots, A_k$  are attributes of the relation  $R$ .
- **Semantics:** Key:  $R[A_1, \dots, A_k]$  is satisfied on a database instance  $D$  if there are no two distinct tuples in  $D(R)$  that agree on attributes  $A_1, \dots, A_k$

# Key Constraints: Example

<u>Student</u>	<b>Id</b>	<b>Surname</b>
	1	Rossi
	2	Rossi
	3	Verdi

The above database satisfies the key constraint:

**key:** Student[Id]

While it does not satisfy the key constraint:

**key:** Student[Surname]

# Inclusion Constraints

- **Intuition:** An **inclusion constraint** imposes to the values on a (non-empty) sequence X of attributes of a relation R to occur as values in a specific sequence Y of attributes of a different relation S.
- **Syntax:**  $R(X) \subseteq S(Y)$   
Where X and Y are non-empty sequences of both **n** attributes of R and S, respectively.
- **Semantics:**  $R(\{i_1, i_2, \dots, i_n\}) \subseteq S(\{j_1, j_2, \dots, j_n\})$  is satisfied in an instance D if for each tuple  $t_1$  in  $D(R)$  there exists a tuple  $t_2$  in  $D(S)$  such that  $t_1[i_k] = t_2[j_k]$ , for each  $k$  in  $\{1, \dots, n\}$

# Inclusion Constraints: Example

<u>Student</u>	<u>Id</u>	<u>Surname</u>
1	Rossi	
2	Rossi	
3	Verdi	

<u>Grades</u>	<u>Student</u>	<u>Exam</u>	<u>Exam</u>
1	Math101	A	
2	CS101	B	
2	CS102	C	

The above database satisfies the inclusion constraint:

$$\text{Grades}[\text{Student}] \subseteq \text{Student}[\text{Id}]$$

While it does not satisfy the inclusion constraint:

$$\text{Student}[\text{Id}] \subseteq \text{Grades}[\text{Student}]$$

# Database Queries

- **What is a database query?**
- **Intuitively:** a way to extract information from a database.
  - Expressed in some language (SQL?)
  - Each expression has an associated meaning, i.e., the result of evaluating the query.
- How do we formally define database queries?

# Database Queries – Formal Definition

- Assume sets of database  $\mathcal{D}$  and a set of relations  $\mathcal{R}$ .
- A **database query** is a function  $q: \mathcal{D} \rightarrow \mathcal{R}$ .
  - Intuitively, given a  $D \in \mathcal{D}$ ,  $q(D) \in \mathcal{R}$  is the relation corresponding to the set of answers for  $q$  over  $D$ .
- This definition is general enough to encompass all possible queries.

# Query Languages

We need to consider queries that we **can actually handle**.

- We need to be able to write them down in a compact way.
- We need to clearly understand the complexity of evaluating queries.

A **query language** is a set of “syntactic” objects usually called *query expressions* or simply *queries*.

Query expressions have an attached meaning.

- Procedurally (relational algebra)
- Declaratively (relational calculus)

# **Logic And Databases**

# Logical Formulae and Logical Theories

- We can use logical formulae in two essential ways
  1. We can evaluate the value of a formula over an interpretation
    - Given an assignment, a formula is either true or false in an interpretation.
  2. We can use formulae to describe a set of “correct” interpretations.
    - All those that satisfy the formulae (independently from the chosen assignment).
    - In this case, we talk about logical theories.
    - Interpretations satisfying a theory represent valid possible worlds.
- Logical theories and logical formulae are a staple of databases.
  - Can you guess why?

# Databases and First-Order Logic

We can see a database instance  $D$  as a (finite) interpretation  $I$ .

- Our **domain** is the set  $\text{dom}(D)$  of attribute values
- Our **alphabet** is formed by:
  - the set **Rels** of relational predicates  $R_{/n}$
  - the set **Cons** of constants (we have a function  $c$  with arity 0 (constant) for each element  $c$  in **dom**).

We use **databases as interpretations**. Hence, given a database  $D$  over a schema  $S = \{R_1, \dots, R_n\}$ , we implicitly see  $D$  as the interpretation  $I = (\text{dom}(D), \cdot^I)$ , where:

- $R^I = D(R)$  for each  $R \in S$  and  $R^I = \emptyset$  for each  $R \notin S$
- $c^I = c$  for each  $c \in \text{Cons}$  (each constant is interpreted into itself)

# Exercise

Write the interpretation that represents the following database.

**Products**

Code	Color
P001	Red
P002	Green
P003	Blue

**Customers**

Code	Name
C001	Eryn
C002	Dylan
C003	Cora

**Orders**

Code	Customer	Product	Quantity
O001	C001	P001	4
O002	C002	P002	2
O003	C001	P001	3

## Exercise 1: Solution

- **Rels** = {*Products*, *Customers*, *Orders* ...}
- **Cons** = {*P001*, *P002*, *P003*, *Red*, *Green*, *Blue*, *C001*, *C002*, *C003*, *Eryn*,  
*Dylan*, *Cora*, *O001*, *O002*, *O003*, *4*, *3* ...}
- $\Delta^I = \{P001, P002, P003, Red, Green, Blue, C001, C002, C003, Eryn, Dylan, Cora, O001, O002, O003, 4, 3\}$
- $Products^I = \{(P001, Red), (P002, Green), (P003, Blue)\}$
- $Customers^I = \{(C001, Eryn), (C002, Dylan), (C003, Cora)\}$
- $Orders^I = \{(O001, C001, P001, 4), (O002, C002, P002, 4), (O001, C001, P001, 3)\}$
- $c^I = c$  for each  $c \in \text{Cons}$

# FOL As a Query Language

# Open and Closed Formulae

Assume a FOL formula  $\varphi$ .

- A variable  $x$  occurring in  $\varphi$  is **bound** if it occurs in the scope of an existential quantifier. A variable is **free** if it is not bound.
- An **open** formula is a FOL formula **with** free variables.
- A **closed** formula (sentence) is a FOL formula **without** free variables.

Closed formulae do not need assignments to be evaluated.

- For such formulae, we write  $I \models \varphi$  instead of  $I, \alpha \models \varphi$  (when this causes no confusion)

**Open formulae require assignments to be evaluated (because of the free variables), so they are the perfect candidate for a query language!**

# Exercise

Which of the following is an open formula?

1.  $\exists x. \forall y. (P(x, y) \wedge R(y, z))$
2.  $\exists x. \forall y. (P(x, y) \wedge \exists z. (R(y, z)))$
3.  $\exists x. \forall y. \exists z. (P(x, y) \wedge R(y, z))$
4.  $\exists x. ((\forall y. \exists z. P(x, y)) \wedge R(x, z))$

## Exercise: Solution

Which of the following is an open formula?

1.  $\exists x. \forall y. (P(x, y) \wedge R(y, \textcolor{red}{z}))$ 
  - **Yes**
2.  $\exists x. \forall y. (P(x, y) \wedge \exists z. (R(y, z)))$ 
  - **No**
3.  $\exists x. \forall y. \exists z. (P(x, y) \wedge R(y, z))$ 
  - **No**
4.  $\exists x. ((\forall y. \exists z. P(x, y)) \wedge R(x, \textcolor{red}{z}))$ 
  - **Yes**

## FOL Queries

- Given a formula  $\varphi$ , we write  $\varphi(x_1, \dots, x_n)$  to denote the fact that  $x_1, \dots, x_n$  are the free variables of  $\varphi$ .
  - Observe that this notation implicitly give us an ordering over  $x_1, \dots, x_n$
  - We assume that no function of arity greater than 0 occurs in  $\varphi$ .
- **Definition:** A FOL Query  $q$  is an expression of the form
$$q = \{(x_1, \dots, x_n) \mid \varphi(x_1, \dots, x_n)\}$$
- Given a database schema  $\mathbf{S}$ , we say that a FOL query  $q$  as above is a query **over  $\mathbf{S}$**  if  $\varphi$  uses only relational predicates in  $\mathbf{S}$ .

## FOL Queries: Answers

- Assume a FOL Query  $q = \{(x_1, \dots, x_n) \mid \varphi(x_1, \dots, x_n)\}$  and an interpretation  $I$
- Given  $\varphi(x_1, \dots, x_n)$ , we use  $\langle c_1, \dots, c_n \rangle$  to denote an assignment  $\alpha$  such that  $\alpha(x_i) = c_i$ , for  $i = 1, \dots, n$
- **Definition:** The **answers for  $q$  over  $I$**  (written  $q(I)$ ) are defined as follows

$$q(I) = \{(c_1, \dots, c_n) \mid I, \langle c_1, \dots, c_n \rangle \models \varphi(x_1, \dots, x_n)\}$$

# FOL Boolean Queries

- A **Boolean FOL query  $q$**  is a FOL query of the form  $\{(\ ) | \varphi\}$ .
  - In other words, a Boolean FOL query is defined using a **closed** formula.
- The **answers for  $q$  over  $I$**  (written  $q^I$ ) are defined as follows
$$\{(\ ) | I, \langle \rangle \models \varphi(x_1, \dots, x_n)\}$$
- **Observe:** if  $q$  is Boolean, then  $q(I) = \emptyset$  or  $q(I) = \{(\ )\}$ , i.e.,  $q(I)$  is either empty or contains only the empty tuple.
  - We understand the former as **false** and the latter as **true**.

# Exercise

Consider the following database D and the FOL query:

$$q = \{(x) | \exists y. \exists z. (Customers(y, x) \wedge Orders(z, y, P002))\}$$

**Products**

Code	Color
P001	Red
P002	Green
P003	Blue

**Customers**

Code	Name
C001	Eryn
C002	Dylan
C003	Cora

**Orders**

Code	Customer	Product
O001	C001	P001
O002	C002	P002
O003	C001	P001

- Write an equivalent SQL query.
- What are the answers of such query?

## Exercise: Solution

- $q = \{(x) | \exists y. \exists z. (Customers(y, x) \wedge Orders(z, y, P002))\}$
- ```
SELECT Customers.name
FROM Customers, Orders
WHERE Customers.Code = Orders.Customer and
      Orders.Product = "P002"
```

- $q(D) = \{(Dylan)\}$

| Name  |
|-------|
| Dylan |

# Complexity of FOL Queries

# Computational complexity – Recap

Computational complexity theory aims to classify computational problems with respect to the resources (space and time) they use.

We are interested in class of **decision problems**, i.e., problems that provide a “**yes**” or a “**no**” answer based on their input.

**Example:** given a string  $s$  and a language  $L$  decide if  $s$  belongs to  $L$

# Evaluating FOL Queries

Evaluating FOL queries over finite interpretations is **decidable**.

- **Query Answering Problem** (Recognition): Given a finite interpretation  $I$ , FOL formula  $\varphi(x_1, \dots, x_n)$ , and a tuple  $\langle c_1, \dots, c_n \rangle$ , check whether  $I, \langle c_1, \dots, c_n \rangle \models \varphi(x_1, \dots, x_n)$ .

**What is the complexity of Query Answering?**

# Evaluating FOL Queries

- Query Answering Problem (Recognition): Given a finite interpretation  $I$ , FOL formula  $\varphi(x_1, \dots, x_n)$ , and a tuple  $\langle c_1, \dots, c_n \rangle$ , check whether  $I, \langle c_1, \dots, c_n \rangle \models \varphi(x_1, \dots, x_n)$ .
- We can define an algorithm to solve the Query Answering Problem
  - Runs in  $O(|I| + |\varphi|^{|I|})$  time.
  - Requires  $O(|I| \cdot |\varphi|)$  space.
- Therefore, the algorithm is exponential. However:
  - Exponential in the **size of the query** (usually small)
  - Polynomial in the **size of the database** (usually large)

# Different Complexities

- **Data Complexity:** The complexity of the problem for a **fixed query**.
- **Query Complexity:** The complexity of the problem for a **fixed interpretation** (so **fixed database**).
- **Combined Complexity:** The complexity of the **whole problem** (query + interpretation)

# Complexities of FOL Query Answering

- **Thm:** Query Answering is PSPACE-complete in Combined Complexity.
- **Corollary:** Query Answering is in EXPTIME in Combined Complexity.
  
- **Thm:** Query Answering is in LOGSPACE in Data Complexity.
- **Corollary:** Query Answering is in PTIME in Data Complexity.
  
- **Thm:** Query Answering is PSPACE-complete in Query Complexity.
- **Corollary:** Query Answering is in EXPTIME in Query Complexity.

# Conjunctive Query

- A FOL formula  $\varphi(x_1, \dots, x_n)$  is an existential conjunction if it is of the form

$$\exists y_1, \dots, \exists y_n P_1(\bar{z}_1) \wedge \dots \wedge P_k(\bar{z}_k)$$

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Conjunctive Query** if  $\varphi(x_1, \dots, x_n)$  is an existential conjunction.

# Union of Conjunctive Queries

- A formula  $\varphi(x_1, \dots, x_n)$  is Union of Existential Conjunctions if it is of the form

$$\bigvee_i \psi_i(x_1, \dots, x_n)$$

where each  $\psi_i$  is an existential conjunction. **Observe:** same free variables.

- A FOL query  $\{(x_1, \dots, x_n) \mid \varphi(x_1, \dots, x_n)\}$  is a **Union of Conjunctive Queries** if  $\varphi(x_1, \dots, x_n)$  is a Union of Existential Conjunctions.

# Complexities of UCQ Answering

- **Thm:** Query Answering is NP-complete in Combined Complexity for UCQs
- **Thm:** Query Answering is in LOGSPACE in Data Complexity.
- **Corollary:** Query Answering is in PTIME in Data Complexity.
- **Thm:** Query Answering is NP-complete in Query Complexity for UCQs.

# **FOL Queries Exercises**

# Exercise

Consider once again the following database

**Products**

| Code | Color |
|------|-------|
| P001 | Red   |
| P002 | Green |
| P003 | Blue  |

**Customers**

| Code | Name  |
|------|-------|
| C001 | Eryn  |
| C002 | Dylan |
| C003 | Cora  |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | P001    |
| O002 | C002     | P002    |
| O003 | C001     | P001    |

# Interpretation

- **Rels** = {*Products*, *Customers*, *Orders*}
- **Cons** = {P001, P002, P003, Red, Green, Blue, C001, C002, C003, Eryn, Dylan, Cora, O001, O002, O003}
- $\Delta^I = \{P001, P002, P003, Red, Green, Blue, C001, C002, C003, Eryn, Dylan, Cora, O001, O002, O003\}$
- $Products^I = \{(P001, Red), (P002, Green), (P003, Blue)\}$
- $Customers^I = \{(C001, Eryn), (C002, Dylan), (C003, Cora)\}$
- $Orders^I = \{(O001, C001, P001), (O002, C002, P002), (O001, C001, P001)\}$
- $c^I = c$  for each  $c \in \text{Cons}$

## Example

Write a FOL query that returns the codes of the customers who ordered a red product.

## Example

Write a FOL query that returns the codes of the customers who ordered a red product.

$$\{(y) | \exists x. \exists z. \textit{Orders}(x, y, z) \wedge \textit{Products}(z, "red")\}$$

## Exercise 3

Write a FOL query, a Relational algebra expression, and an SQL expression for the following query:

"Return the code of those customers who have ordered at least one item"

## Exercise 3 - Solution

Write a FOL query, a Relational algebra expression, and an SQL expression for the following query:

"Return the code of those customers who have ordered at least one item"

- $\{(y) \mid \exists x. \exists z. Orders(x, y, z)\}$
- $PROJ_2(Orders)$
- **SELECT Customer FROM Orders**

## Exercise 4

Write a FOL query, a Relational algebra expression, and an SQL expression for the following query:

"Return the names of those customers who ordered product P001"

## Exercise 4 - Solution

Write a FOL query, a Relational algebra expression, and an SQL expression for the following query:

"Return the names of those customers who ordered product P001"

- $\{(x) \mid \exists y. \exists z. Orders(y, z, P001) \wedge Customers(z, x)\}$
- $PROJ_3(SEL_{1=2}(PROJ_2(SEL_{3="P001"}(Orders)) \times Customers))$
- **SELECT c.name FROM Orders o, Customers c  
WHERE o.Product = "P001" and o.customer = c.code**

## Exercise 5

Write a FOL query, a Relational algebra expression, and an SQL expression for the following query:

"Find the codes of those customers who ordered a red product"

## Exercise 5 - Solution

Write a FOL query, a Relational algebra expression, and an SQL expression for the following query:

"Find the codes of those customers who ordered a red product"

- $\{(y) \mid \exists x. \exists z. Orders(x, y, w) \wedge Products(w, Red)\}$
- $PROJ_2(SEL_{3=4 \wedge 5="Red"} (Orders \times Products))$
- **SELECT Customer FROM Orders o, Products p  
WHERE o.Product = Products.id and  
p.color = "Red"**

## Exercise 6

Write a FOL query, and R.A. expression for the following query:

"Find the codes of customers who ordered a red and a green product"

## Exercise 6 - Solution

Write a FOL query, and R.A. expression for the following query:

"Find the codes of customers who ordered a red and a green product"

- $\{(x) \mid \exists y. \exists z. \exists y'. \exists z'. (Orders(y, x, z) \wedge Products(z, Green)) \wedge (Orders(y', x, z') \wedge Products(z', Red))\}$
- $PROJ_2(SEL_{3=4}((SEL_{5="red"} (Orders \times Products)))) \cap PROJ_2(SEL_{3=4}((SEL_{5="green"} (Orders \times Products))))$

## Exercise 7

Write a FOL query, and a R.A. expression for the following query:

"Find the codes of customers who ordered a red or a green product"

## Exercise 7 - Solution

Write a FOL query, and a R.A. expression for the following query:

"Find the codes of customers who ordered a red or a green product"

- $\{(x) \mid \exists y. \exists z. \exists y'. \exists z'. (Orders(y, x, z) \wedge Products(z, Green)) \vee (Orders(y', x, z') \wedge Products(z', Red))\}$
- $PROJ_2(SEL_{3=4}((SEL_{5="red"}(Orders \times Products))) \cup PROJ_2(SEL_{3=4}((SEL_{5="green"}(Orders \times Products))))$

## Exercise 8

Write a FOL query, and a R.A. expression for the following query:

"Find the code of customers who have ordered the same product twice"

## Exercise 8 - Solution

Write a FOL query, and a R.A. expression for the following query:

"Find the code of customers who have ordered the same product twice"

- $\{(y) \mid \exists x. \exists x'. \exists z. Orders(x, y, z) \wedge Orders(x', y, z) \wedge \neg(x = x')\}$
- $PROJ_2(SEL_{2=5}(SEL_{3=6}(SEL_{1 \neq 4}((Orders \times Orders))))$
- We can avoid the explicit use of  $1 \neq 4$  using difference but the solution is much more involved.

## Exercise 9

Write a FOL query, and a R.A. expression for the following query:

"Find the code of customers who have not ordered a red item"

## Exercise 9 - Solution

Write a FOL query, and a R.A. expression for the following query:

"Find the code of customers who have not ordered a red item"

- **Solution**
  - $\{(y) \mid \exists w. Customers(y, w) \wedge (\neg \exists x. \exists z. (Orders(x, y, z) \wedge Products(z, Red)))\}$
  - $\{(y) \mid \exists w. Customers(y, w) \wedge (\forall x. \forall z. \neg(Orders(x, y, z) \wedge Products(z, Red)))\}$
- **Observe:** the following is wrong. Why?
  - $\{(y) \mid \exists w. Customers(y, w) \wedge \exists x. \exists z. \neg(Orders(x, y, z) \wedge Products(z, "Red"))\}$

## Exercise 10

Write a FOL query and a R.A. expression for the following query:

"Find the code of customers who have ordered all available products"

## Exercise 10 - Solution

Write a FOL query and a R.A. expression for the following query:

"Find the code of customers who have ordered all available products"

-- We assume that all the available products are in Products

- $\{(x) | \exists n. Customers(x, n) \wedge \forall y. \forall z. Products(y, z) \rightarrow \exists w. Orders(w, x, y)\}$
- $PROJ_1(Customers) - PROJ_2(e)$
- $e = (PROJ_1(Products) \times PROJ_1(Customer)) - PROJ_{2,3}(Orders)$

## Exercise 11

Write a FOL query, and a R.A. expression for the following query:

"Find the names of customers who have ordered all available green products"

## Exercise 11 - Solution

Write a FOL query, and a R.A. expression for the following query:

"Find the names of customers who have ordered all available green products"

- $\{(x) | \exists z. Customers(z, x) \wedge \forall y. Products(y, Green) \rightarrow \exists w. Orders(w, x, y)\}$
- $PROJ_1(Customers) - PROJ_2(e)$
- $e = (PROJ_1(SEL_{2="green"}(Products)) \times PROJ_1(Customer)) - PROJ_{2,3}(Orders)$

# **FOL Integrity Constraints Exercises**

# FOL Constraints

- A set of first-order formulae can define integrity constraints.
  - A database satisfies the set of constraints if it is a model of the associated logical theory
- Usually, constraints are expressed as **closed formulae**.
- FOL can express standard constraints in the relational model.

## FOL Constraints -- Examples

Assume relations  $R_{/3}$  and  $S_{/3}$ . Write in FOL the following constraints.

- $R(1,2) \subseteq S(2,3)$
- $R(1,2)$  is a key of  $R_{/3}$

## FOL Constraints -- Examples

Assume relations  $R_{/3}$  and  $S_{/3}$ . Write in FOL the following constraints.

- $R(1,2) \subseteq S(2,3)$ 
  - $\forall x. \forall y. \forall z. R(x,y,z) \rightarrow \exists w. S(w,x,y)$
- $R(1,2)$  is a key of  $R_{/3}$ 
  - $\forall x. \forall y. \forall z. \forall w. R(x,y,z) \wedge R(x,y,w) \rightarrow (z = w)$

# Exercise 12

Consider once again the following database

**Products**

| Code | Color |
|------|-------|
| P001 | Red   |
| P002 | Green |
| P003 | Blue  |

**Customers**

| Code | Name  |
|------|-------|
| C001 | Eryn  |
| C002 | Dylan |
| C003 | Cora  |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | P001    |
| O002 | C002     | P002    |
| O003 | C001     | P001    |

# Interpretation

- **Rels** = {*Products*, *Customers*, *Orders*}
- **Cons** = {P001, P002, P003, Red, Green, Blue, C001, C002, C003, Eryn, Dylan, Cora, O001, O002, O003}
- $\Delta^I = \{P001, P002, P003, Red, Green, Blue, C001, C002, C003, Eryn, Dylan, Cora, O001, O002, O003\}$
- $Products^I = \{(P001, Red), (P002, Green), (P003, Blue)\}$
- $Customers^I = \{(C001, Eryn), (C002, Dylan), (C003, Cora)\}$
- $Orders^I = \{(O001, C001, P001), (O002, C002, P002), (O001, C001, P001)\}$
- $c^I = c$  for each  $c \in \text{Cons}$

## Exercise 12

Write in first-order logic all the integrity constraints you think are reasonable for the database above.

# Exercise 12 - Solution

## Foreign Keys

- $\forall x. \forall y. \forall z. Orders(x, y, z) \rightarrow \exists w Customers(y, w)$  (**Orders(2) ⊆ Customers(1)**)
- $\forall x. \forall y. \forall z. Orders(x, y, z) \rightarrow \exists w Products(z, w)$  (**Orders(3) ⊆ Products(1)**)

## Key Constraints

- $\forall x. \forall y. \forall z. \forall y'. \forall z'. Orders(x, y, z) \wedge Orders(x, y', z') \rightarrow y = y' \wedge z = z'$
- $\forall x. \forall y. \forall y'. Customers(x, y) \wedge Customers(x, y') \rightarrow y = y'$
- $\forall x. \forall y. \forall y'. Products(x, y) \wedge Products(x, y') \rightarrow y = y'$

## Other constraints

- $\forall x. \forall y. Customers(x, y) \rightarrow \exists z. \exists w Orders(z, x, w)$
- Every customer made at least one order

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

# Information Integration Systems

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
- 4. Information Integration Systems (IIS)**
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
- 9. Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# **Problem Definition**

# Information Integration

**Information integration** is the process of **reconciling** information coming from **multiple, autonomous**, and **heterogeneous** sources.

# Information Management

What are the tasks of interest?

- **Query.**
  - Answering user-specified queries over the reconciled view of the information at the sources.
- **Update, Delete, and Insert.**
  - Updating, deleting, and inserting information using the reconciled view of the information at the sources.

# Physical vs Conceptual Heterogeneity

- **Physical Heterogeneity:** Information is represented using different formats. Datatypes and data models of the information sources may differ.
- **Conceptual Heterogeneity:** The information sources describe the world from different perspectives.

# Physical Heterogeneity: Examples

- **Legacy databases**
  - Hierarchical model
  - Network model
- **Post-Relational**
  - Object Databases
  - Multi-value DBs
  - Graphs
- **SQL database.**
  - We can use SQL or a dialect thereof.
  - Relational Data Model.
- **NoSQL Databases**
  - Key-value stores
  - Big Tables
  - Documents (Json or the like)

# Conceptual Heterogeneity: Example

| Name     | DOB        | Residency      |
|----------|------------|----------------|
| Amber A. | 10/20/2030 | New Amsterdam  |
| Rose B.  | 10/20/1930 | Constantinople |

| ID   | Customer Name | Product |
|------|---------------|---------|
| O001 | Amber A.      | P001    |
| O002 | Dylan B.      | P001    |

**Both databases are relational** (so, no physical heterogeneity)

- What are the persons?
- What's their DOB?

# Contexts of Application

Information Integration has mainly two contexts of application.

- **Intra-organizational.**
  - Different entities of an organization produce and consume information.
  - How can they communicate?
- **Inter-organizational.**
  - Different organizations produce and consume information.
  - How can they communicate?

# **Classifications**

# Information Integration Systems

An information Integration System is a piece of software that reconcile information coming from multiple sources.

We talk about **data integration tools** to refer to information integration systems that directly deal with data (in the sense of databases)

# Classifications of Information Integration Systems

Information integration systems can be classified according to different characteristics.

- **Scope of the Integration:** What is the extent to which information should be integrated?
- **System Specification:** How is the integration process specified?
- **Information Replication and Timeliness:** What is the end result of the integration?

# **Classifications**

**Scope of the integration**

# Scope of the Integration

According to the scope of the integration process, we classify information integration systems in three categories.

- **Task-Based Systems (one-shot)**: Systems built to perform a single integration task.
- **Service-Based Systems (query based)**: Systems built to perform a specific task (query) over time.
- **Domain Integration System (general purpose)**: Systems built to perform user-specified tasks (queries).

# Task-Based Information Integration Systems

Task-Based systems perform a single specific task.

- Extract specific information from the sources at a specific point in time.

**They are not meant for reuse:**

- Code is written specifically for the task and may be discarded after use.

**Possible applications:**

- Data migration (loading pre-existing data into a new system)
- Research-specific tasks (extracting business insight from data)

# Task-Based Systems: Example 1

## Data Migration Task:

**Data Sources:** customer data from different branches of an acquiree company.

**Task:** Load data into the information system of the acquiring company.

# Task-Based Systems: Example 2

## Research-Specific Task:

**Data Sources:** Occupational data from the EU countries to date.

## Task(s):

- Extract occupancy rate trend for the last century.
- Extract agricultural workers trend over the last century.

# Service-Based Information Integration Systems

Service-Based systems perform a **specific task** over time.

- Extract specific information from the sources whenever is needed.

**Meant for reuse:**

- Service-Based systems may use user interfaces to facilitate interaction.
- Data may be a snapshot taken at a given point in time or refreshed at every query.

**Limited Scope:**

- While task may be repeated, the user have no freedom to choose the task.

**Possible applications:**

- Parametric query execution (content aggregators)

# Service-Based Systems: Example 1

## Content Aggregator:

**Data Sources:** availability and cost of flights from different companies.

**Task:** Search for the cheapest flight for a given destination and date.

## Service-Based Systems: Example 2

**Content Aggregator:**

**Data Sources:** prices of different goods from different sellers.

**Task:** Search for the cheapest product of a kind specified by the user.

# **Domain-Based Information Integration Systems**

Domain-Based I.I.S. allow users to specify queries over the domain.

- Extract specific information from the sources whenever is needed.

## **Meant for reuse and broad scope:**

- Domain-Based systems may use user interfaces to facilitate interaction.
- Data may be a snapshot taken at a given point in time or refreshed at every query.
- Different tasks (queries) can be performed over time.

## **Possible applications:**

- Enterprise System Integration.
- Auditing.

## Domain-Based Systems: Example

**Auditing:**

**Data Sources:** account data from multiple banks.

**Task:** Perform queries to track money laundering activities.

# Domain-Based Information Integration

- **Domain-Based Information Integration** is the problem of providing a **unified view** of a collection of heterogenous and autonomous sources.
- **Essentially, our idea of information integration is domain-base integration.**
- The **unified view** is achieved using a **global (or target) schema**, i.e., a formal definition of what the **result of the integration should look like**.
  - Information at the sources is described by source schema(s), i.e., formal definitions how information looks like at the **sources**.

# Domain-Based Information Integration

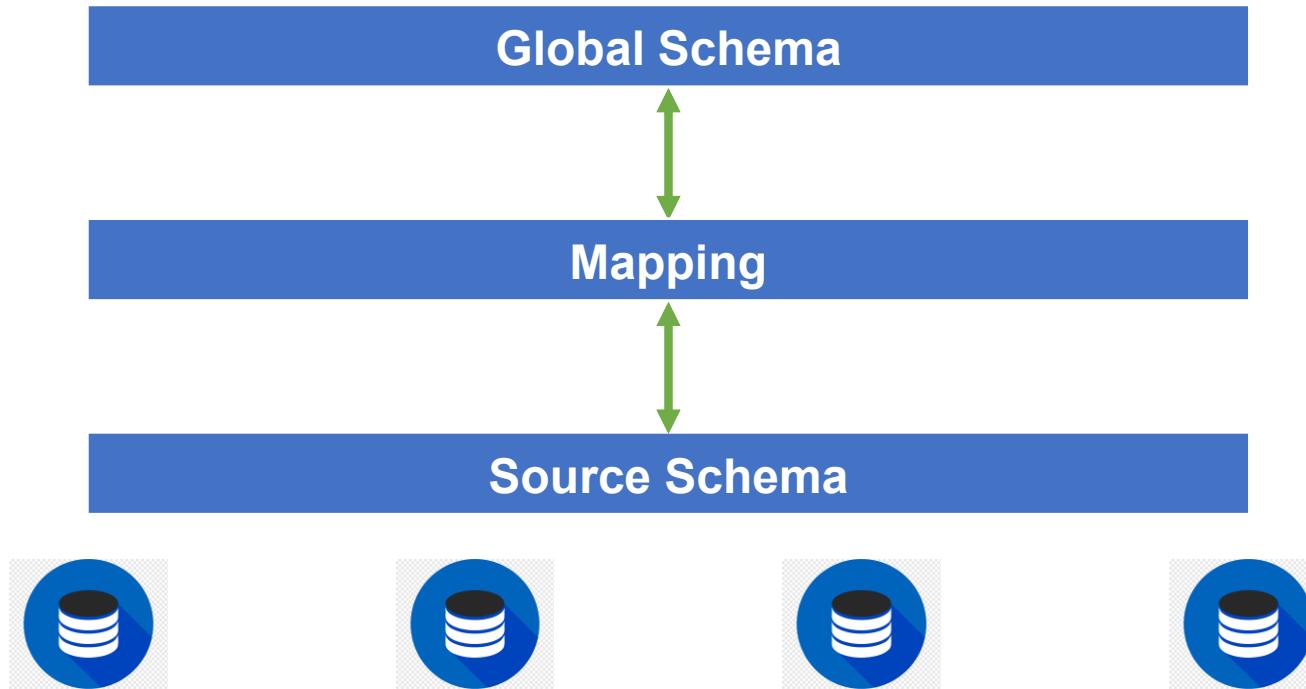
A **mapping layer** describes the **target data** (data in unified view) in terms of the **source data**.

Different approaches are possible

- **Ex 1:** Describing queries over the target schema in terms of queries over the sources.
- **Ex 2:** Sets of logical axioms
- **Ex 3:** Define data transformation processes.

**From now on, we focus on Domain-Based Integration, so we will refer to Domain-Based Integration simply as Information Integration.**

# Information Integration Framework



# **Classifications**

**System Specification (Mapping Specification)**

# Mapping Specification

There are several ways to specify the **mapping** of an information integration system

## Declarative

- The system is defined by specifying **what information** should be extracted
- Often using formal tools such as **first-order logic**.

## Procedural (query-by-query)

- Mapping is defined by specifying the **process that should be used to perform the tasks**.
- Often using programming languages or procedural query languages (relational algebra)

# Data Sources – Example

| Name     | DOB        | Residency      |
|----------|------------|----------------|
| Amber A. | 10/20/2030 | New Amsterdam  |
| Rose B.  | 10/20/1930 | Constantinople |

| ID   | Customer Name | Product |
|------|---------------|---------|
| O001 | Amber A.      | P001    |
| O002 | Dylan B.      | P001    |

**Global Schema:** persons and an id code

# Declarative Mapping Specification: Example

**Source Schema:** Customers(Name, DOB, Residency)  
Purchases(Id, Customer\_Name, Product)

**Target Schema:** Persons(Id, Name)

**Declarative Mapping Definition (FOL):**

- $\forall n, d, r. \text{Customers}(n, d, r) \rightarrow \exists i. \text{Persons}(i, n)$
- $\forall i_p, p, n. \text{Purchases}(i_p, n, p) \rightarrow \exists i. \text{Persons}(i, n)$

# Procedural Mapping Specification: Example 1

**Source Schema:** Customers(Name, DOB, Residency)  
Purchases(Id, Customer\_Name, Product)

**Target Schema:** Persons(Id, Name)

**Procedural Mapping Definition (Relational Algebra):**

- Names:  $\text{Proj}_1(\text{Customers}) \cup \text{Proj}_3(\text{Purchases})$
- A mechanism to assign ids (more than relational algebra)

# Procedural Mapping Specification: Example 2

**Source Schema:** Customers(Name, DOB, Residency)  
Purchases(Id, Customer\_Name, Product)

**Target Schema:** Persons(Id, Name)

## Procedural Mapping Definition (Software Based):

- Software modules written to extract the data from the sources.

# Implementing the Mapping

When mappings are defined **declaratively**, we need an implementation

## Ad-hoc Software

- Build a system from scratch that implements the mapping
- What happens if the mapping evolves?

## Use an information integration tool

- Tools that provide mechanism to specify mapping using **SQL**
- Data can be processed in multiple steps that may include running arbitrary code on it.

# **Classifications**

**Data Replication and Timeliness**

# Data Replication and Timeliness

What is the result of the integration?

- **Materialization**
  - A representation of the result of the integration is loaded from the sources into another database
  - Such representation represents a snapshot of the data at a specific time.
- **Virtualization (Mediator-Based)**
  - No concrete representation of the information is loaded (data are not moved)
  - All tasks are performed **on-the-fly** over the data sources (always up-to-date)

# Materialized Information Integration

In **materialized** data integration, a **representation** of the result of the integration process is loaded from the sources

Such loading process can be either lazy or continuous

- Continuous loading is often used in data warehousing.
- Lazy loading is used to perform specific tasks or to obtain snapshots of the data.

Often performed using specific tools

- Extraction, transformation, and loading (ETL) tools that allow one to specify what data to extract (declaratively) and how to do it (procedurally)

In the literature: **Data Exchange**

# Virtual Information Integration

In **virtual** data integration, all tasks are performed directly at the sources.

- Users' tasks are specified over the **global schema** but data stays at the sources
  - No unnecessary movement of data.
  - No need to update a local representation of the integration process.
  - Does not provide a mechanism for snapshotting data.
- Multiple software components perform the tasks over the sources
  - **Wrappers** are used to access the sources and present data in a standard format, thus solving physical heterogeneity.
  - A **mediator** combines and reconciles information coming from the wrappers, thus solving conceptual heterogeneity.

# **Approaches to Information Integration**

# Approaches to Information Integration

The problem of integrating information from multiple sources is as old as business organizations.

Over the years, different technological approaches have been proposed.

- Some may still be relevant today, depending on the size of the organization.

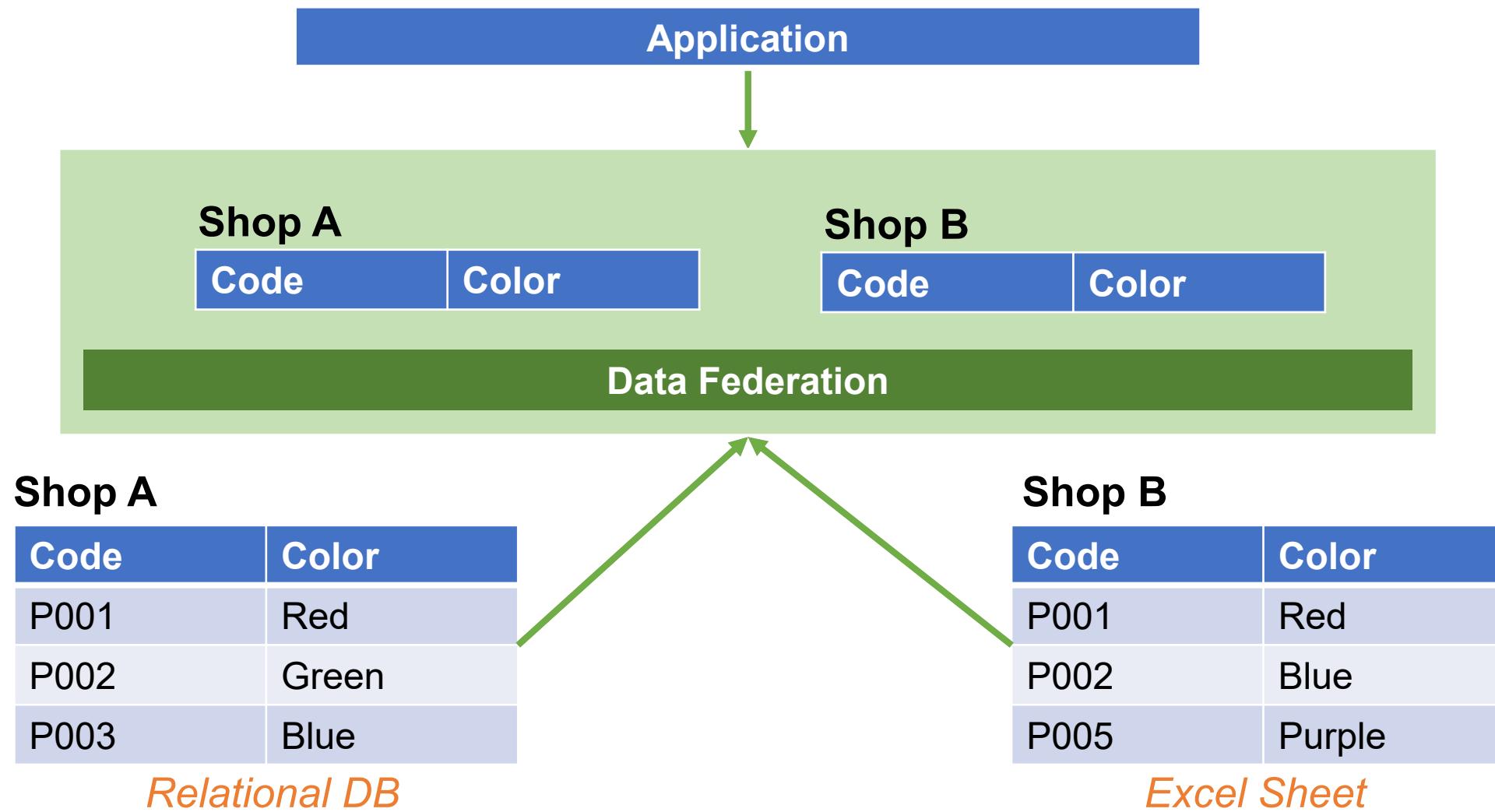
We survey some relevant approaches under the lens of the notions introduced in the previous part of this lecture.

# Data Integration vs Data Federation

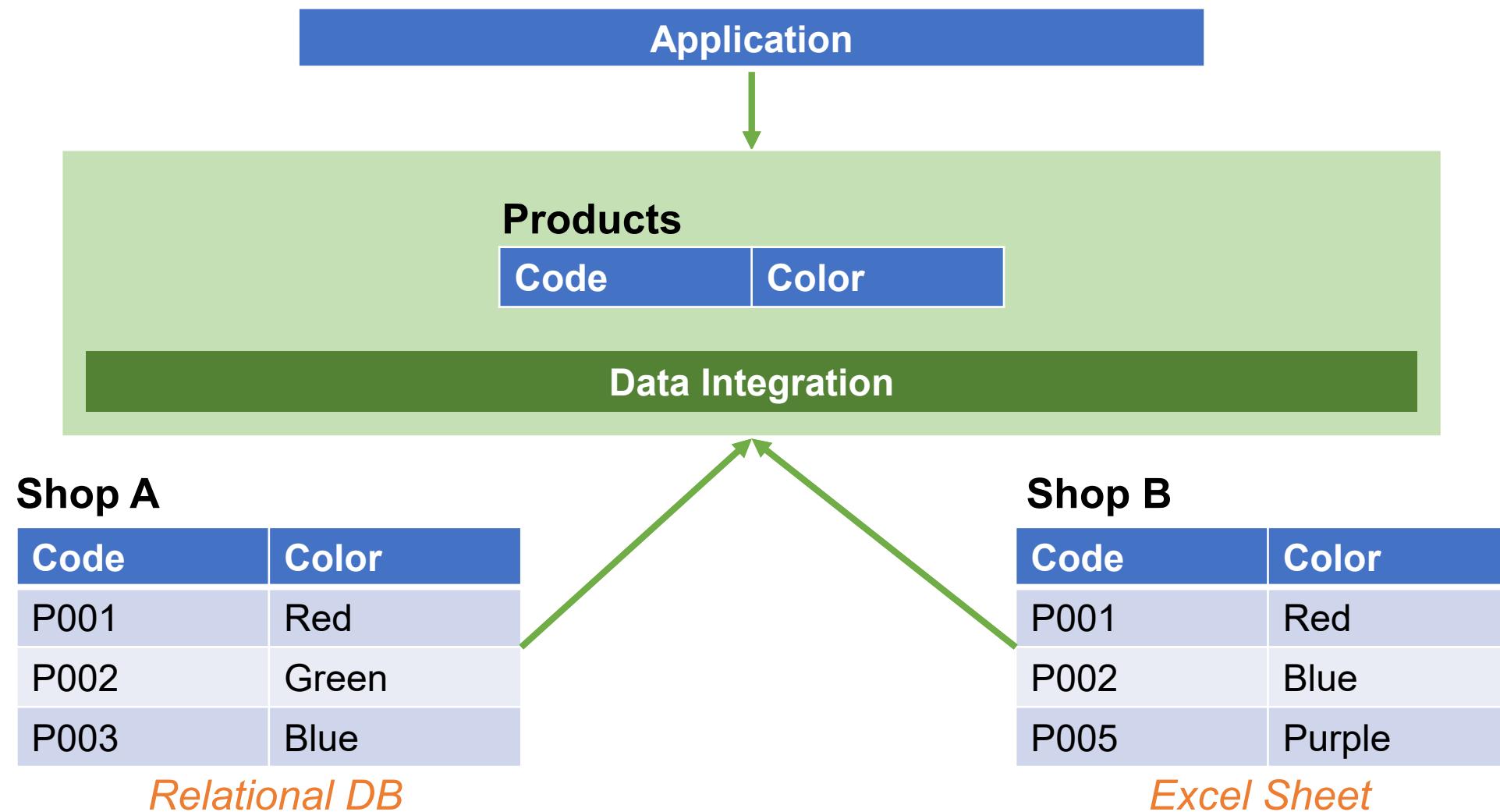
- A common source of confusion is the difference between **data integration** and **data federation** tools.
- **Data federation tools:** provide **access** to data stored in multiple sources.
- **Data integration tools:** reconcile data stored in multiple, autonomous, and heterogeneous sources.

**The difference is Logical (or Conceptual) Transparency** (the way data is stored at the sources does not affect the way the information is presented to the end user)

# Data Federation



# Data Integration



# Achieving Logical Transparency

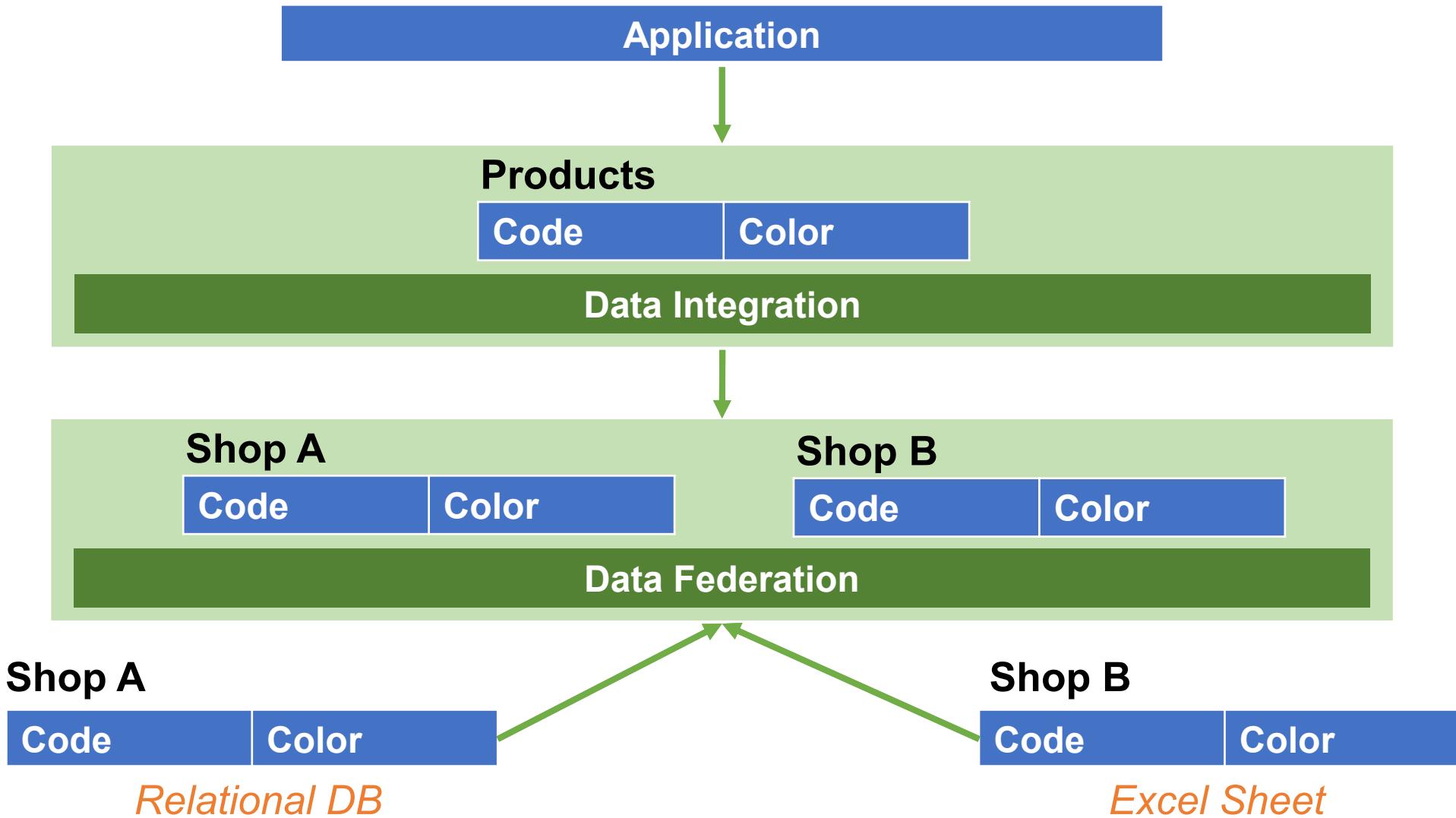
Data Integration tools achieve Logical Transparency using:

- A **Global Schema** providing a view of the scenario of interest independent from the way the **sources** store the data
- A **Mapping Layer** that reconciles the **source schema** with the **global schema**
- Rich formalisms for the specification of **global schemas** and **mappings**.

Data federation and data integration are often orthogonal.

- Data integration tools may be built on top of data federation tools.
- Often, we can use data federation tools to build the **wrappers** of an information integration tool.

# Data Integration + Data Federation



# **Challenges of Information Integration**

# Challenges of Information Integration

Information integration may be a very challenging task both technologically and from a theoretical point of view

In the next slides, we briefly discuss some of the challenges of information integration

This list has no claim of completeness!

Indeed, Many challenges of information integration are domain specific.

# Language-Related Challenges

- Expressive power
  - What formalisms should we use to express **global schema** and **mapping**?
  - Languages too rich may give rise to hard or undecidable tasks
  - Languages too simple may not be enough to define the desired properties.
- Modelling problem
  - Fixed a formalism, what is the best way to capture the intended properties?
- Automatic definitions
  - Can we devise tools to automatically define **mappings**?

# Data-Management Challenges

- Efficient Query Processing
  - Depending on how conceptually different are the **sources**, information integration systems may need to specify very complex queries.
  - How can we **speed up** the computation of such queries?
  - How can we materialize the right amount of information?
- Information Management
  - How can we updates, deletions, and insertions expressed over the **global schema**?
  - What does it mean to update the **sources** w.r.t. the **global schema**?
- Data source wrapping
  - How can we efficiently define wrappers for data **sources**?
  - Can we do it automatically, i.e., can we automatically discover the right types?

# Information Integration Challenges

- Virtual Data Integration
  - How do we translate a query over the **global schema** into a query for the **sources**?
- Materialized Data Integration
  - What kind of representation should we load from the **sources**?
- Global Schema Constraints
  - Can we define further requirements over the **global schema**, independently from the **mapping**?

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Logical Formalization of Information Integration Systems

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. **Logical formalization of IISs**
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. **Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# Logical Formalization

# Formalization of Information Integration Systems

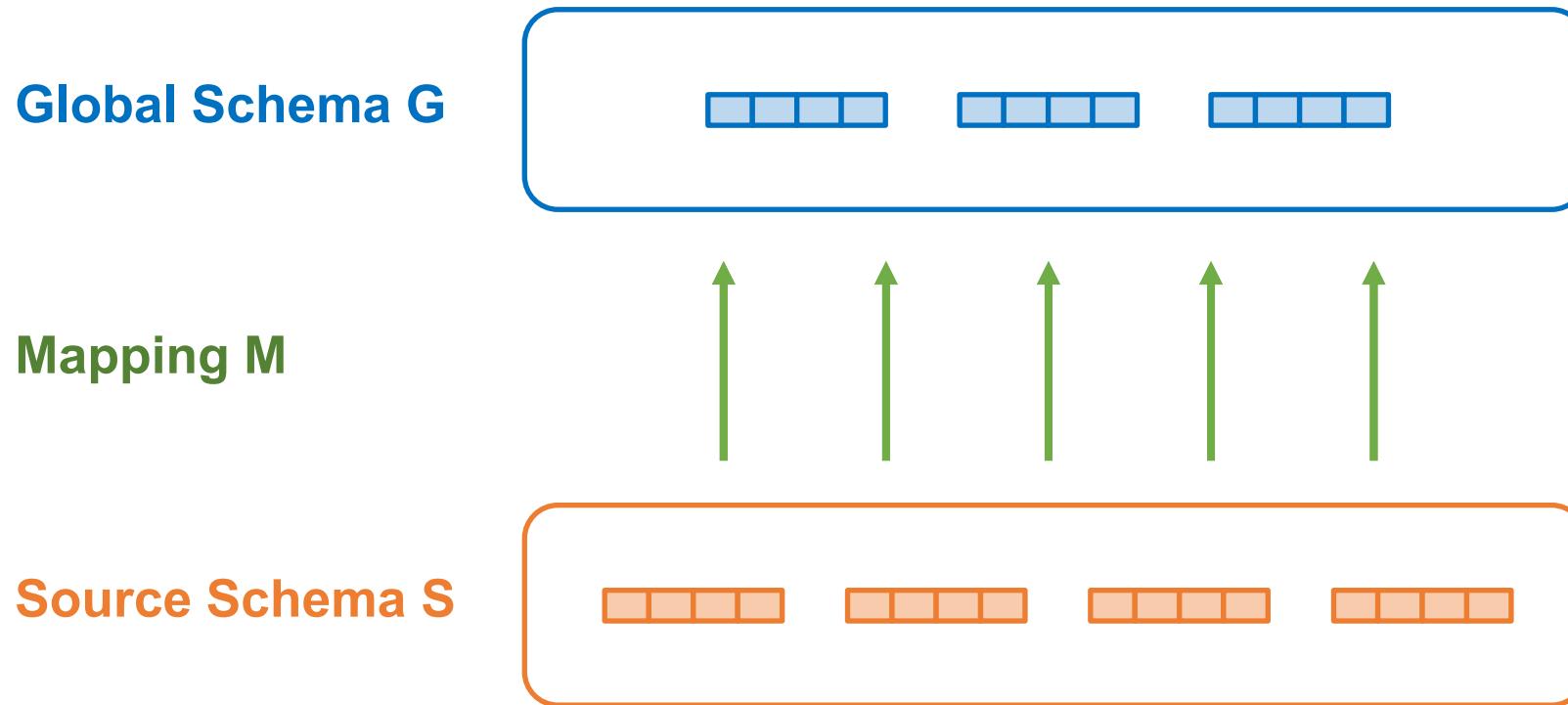
- The architecture of an Information Integration Systems (I.I.S.) can be defined using **First-Order Logic**
  - However, we need to make some simplifying assumptions
- Data sources are represented as one single relational database
  1. No syntactic heterogeneity between sources
  2. A single logical theory (**source schema**) describes valid data sources
  3. Source schemas may contain constraints (FOL sentences)
- The result of the integration is a relational database.
  1. No syntactic heterogeneity between sources and the result of integration
  2. A single logical theory (**global schema**) describes the result of the integration
  3. Global schemas may contain constraints (FOL sentences)

# Information Integration Specification - Syntax

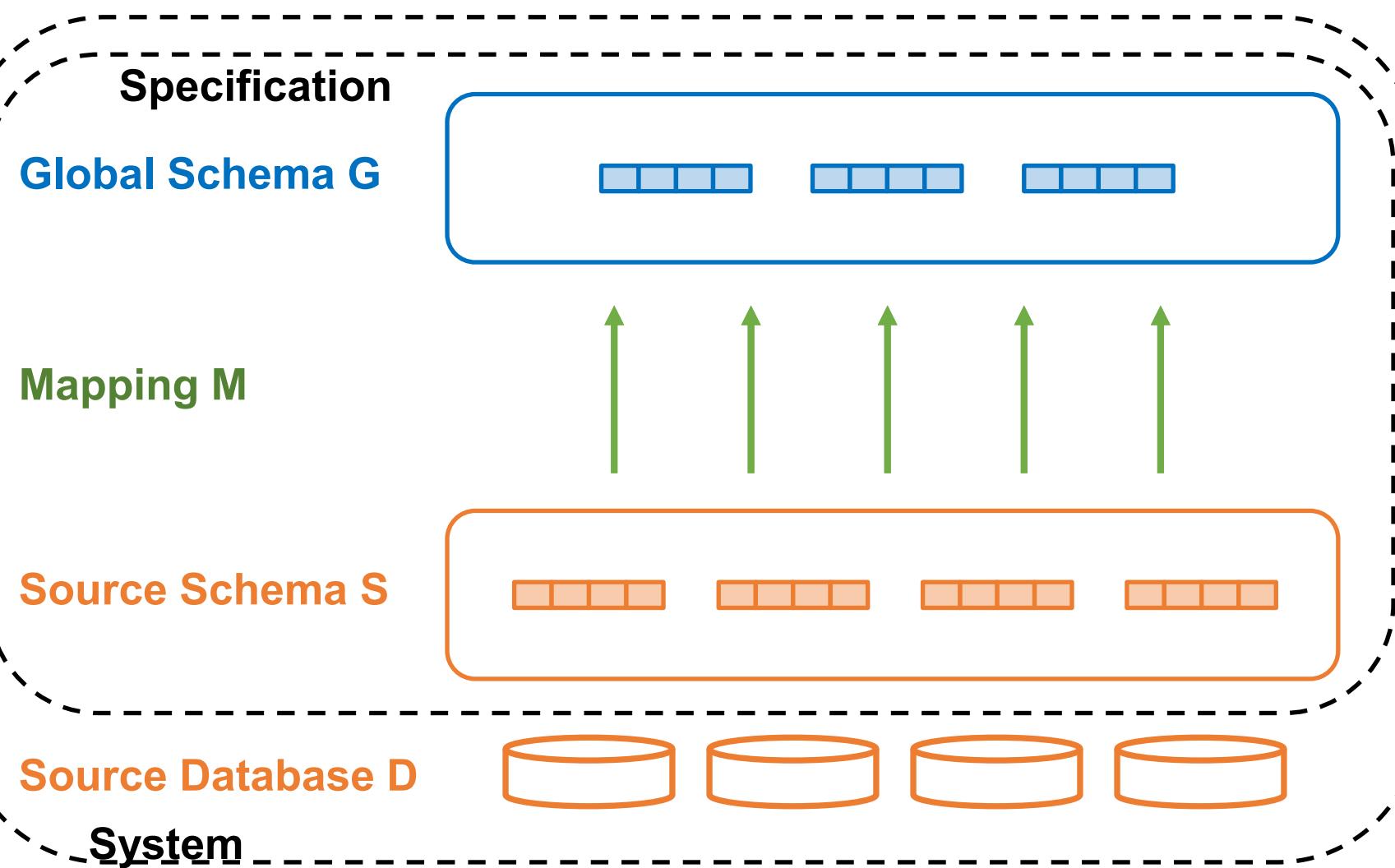
An information integration specification is a triple  $J = \langle G, M, S \rangle$ , where:

- **G** is the **global schema**
  - A relational database schema that describes how data resulting from the integration process should look like
  - A database **B** for **G** will be called **global database**, i.e., we assume that  $B \models G$
- **S** is the **source schema**
  - A relational database schema describing the data sources such that  $G \cap S = \emptyset$
  - A database **D** for **S** will be called **source database**, i.e., we assume that  $D \models S$
- **M** is the **mapping** between **source schema S** and **global schema G**
  - A set of mapping assertions  $m = \langle q_S, q_G \rangle$ , where  $q_S$  is a FOL query over **S** and  $q_G$  is a FOL query over **G**.  $q_S$  and  $q_G$  have the same arity!

# Information Integration Specification



# Information Integration System - Syntax



An information integration **system** is a pair  $\langle J, D \rangle$ , where:

- $J = \langle G, M, S \rangle$  is an information integration **specification**
- $D$  is a **source database** for  $S$ , i.e.,  $D \models S$

# Information Integration System - Semantics

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$

- A global database  $B$  **satisfies** a mapping assertion  $m = \langle q_S, q_G \rangle$  in  $M$  with respect to  $D$ , denoted by  $(B, D) \models m$ , if  $q_S(D) \subseteq q_G(B)$   
**(sound source assumption)**
- A global database  $B$  **satisfies** the mapping  $M$  with respect to  $D$ , denoted by  $(B, D) \models M$ , if  $(B, D) \models m$  for each  $m \in M$

Intuitively, the semantics of  $\langle J, D \rangle$  is the set of **global databases**, i.e., is a database for  $G$ , that satisfy the **mapping** w.r.t. to the **source database**.  
Formally:

$$\text{sem}(J, D) = \{ B \mid B \text{ is a global database s.t. } (B, D) \models M \}$$

# Source Schema and Database

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Purchases

| ID   | Customer Name | Product |
|------|---------------|---------|
| O001 | Amber         | P001    |
| O002 | Dylan         | P001    |

# Global Schema

Person

| Id | Name |
|----|------|
|----|------|

# Example of Information Integration System

Assume  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  with:

- $\mathbf{G} = \langle Person_{/2} \rangle$
- $\mathbf{S} = \langle Purchases_{/3}, Customers_{/3} \rangle$
- $\mathbf{M} = \{m_1, m_2\}$  where:
  - $m_1 = \langle \{(n) | \exists id. \exists p. Purchases(id, n, p)\}, \{(n) | \exists i. Person(i, n)\} \rangle$
  - $m_2 = \langle \{(n) | \exists d. \exists r. Customers(n, d, r)\}, \{(n) | \exists i. Person(i, n)\} \rangle$

Assume a **source database D** as defined previously

See exercises in the next slides!

# Exercise 1

Provide a global database over  $\mathbf{G} = \{Person_{/2}\}$  that belongs to  $\text{sem}(\mathbf{J}, \mathbf{D})$

---

$\mathbf{M} = \{m_1, m_2\}$  where:

- $m_1 = \langle \{(n) | \exists id. \exists p. Purchases(id, n, p)\}, \{(n) | \exists i. Person(i, n)\} \rangle$
- $m_2 = \langle \{(n) | \exists d. \exists r. Customers(n, d, r)\}, \{(n) | \exists i. Person(i, n)\} \rangle$

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Purchases

| ID   | Customer Name | Product |
|------|---------------|---------|
| O001 | Amber         | P001    |
| O002 | Dylan         | P001    |

# Exercise 1 - Solution

Provide a global database over  $\mathbf{G} = \{Person_2\}$  that belongs to  $\text{sem}(\mathbf{J}, \mathbf{D})$

---

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |
| 3  | Dylan |

## Exercise 2

Which of the following global databases belong to  $\text{sem}(\mathbf{J}, \mathbf{D})$ ? Why?

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |
| 3  | Dylan |
| 2  | Dylan |

B1

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Jonny |
| 3  | Dylan |

B2

Person

| Id | Name  |
|----|-------|
| a  | Amber |
| b  | Rose  |
| c  | Dylan |
| d  | Jonny |

B3

## Exercise 2 - Solution

Which of the following global databases belong to  $\text{sem}(\mathbf{J}, \mathbf{D})$ ? Why?

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |
| 3  | Dylan |
| 2  | Dylan |

B1

**B1:** Yes!

**B2:** No! Rose is missing

**B3:** Yes!

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Jonny |
| 3  | Dylan |

B2

Person

| Id | Name  |
|----|-------|
| a  | Amber |
| b  | Rose  |
| c  | Dylan |
| d  | Jonny |

B3

# Querying an Information Integration System

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$

Querying  $\langle J, D \rangle$  simply means posing queries over the **global schema  $G$**  of the information integration specification  $J = \langle G, M, S \rangle$

Intuitively, the **certain answers** for a FOL query  $q$  w.r.t.  $\langle J, D \rangle$  are those tuples of constants that occur in the answers for  $q$  over **every possible global database** in the semantics of  $\langle J, D \rangle$ . Formally:

$$\text{cert}(q, J, D) = \{(c_1, \dots, c_n) \mid (c_1, \dots, c_n) \in q(B) \text{ for each } B \in \text{sem}(J, D)\}$$

## Exercise 3

Consider the previous information integration system  $\langle \mathbf{J}, \mathbf{D} \rangle$ , and consider the following queries:

$$q_1 = \{ () \mid \exists n. \exists i. \text{Person}(i, n) \}$$

$$q_2 = \{ (n) \mid \exists i. \text{Person}(i, n) \}$$

$$q_3 = \{ (i) \mid \exists n. \text{Person}(i, n) \}$$

$$q_4 = \{ (i, n) \mid \text{Person}(i, n) \}$$

Compute  $\text{cert}(q_1, \mathbf{J}, \mathbf{D})$ ,  $\text{cert}(q_2, \mathbf{J}, \mathbf{D})$ ,  $\text{cert}(q_3, \mathbf{J}, \mathbf{D})$ , and  $\text{cert}(q_4, \mathbf{J}, \mathbf{D})$ .

## Exercise 3 - Solution

Consider the previous information integration system  $\langle J, D \rangle$ , and consider the following queries:

$$q_1 = \{ () \mid \exists n. \exists i. Person(i, n) \}. cert(q_1, J, D) = \{()\} \text{ (true)}$$

$$q_2 = \{ (n) \mid \exists i. Person(i, n) \}. cert(q_2, J, D) = \{(Amber), (Rose), (Dylan)\}$$

$$q_3 = \{ (i) \mid \exists n. Person(i, n) \}. cert(q_3, J, D) = \emptyset$$

$$q_4 = \{ (i, n) \mid Person(i, n) \}. cert(q_4, J, D) = \emptyset$$

For  $q_1$  and  $q_2$ , note that Amber, Rose, and Dylan must occur in all global databases  $B \in sem(J, D)$

For  $q_3$  and  $q_4$ , consider the global databases  $B1 \in sem(J, D)$  and  $B3 \in sem(J, D)$  given before. Note that  $q_i(B1) \cap q_i(B3) = \emptyset$  for both  $i=3$  and  $i=4$

# **Complexity of Query Answering**

# Complexity of Query Answering

- Thm: Let  $J = \langle G, M, S \rangle$  be an information integration specification,  $D$  be a **source database** for  $S$ , and  $q$  be a FOL query over  $G$ . Computing  $\text{cert}(q, J, D)$  is **undecidable**.
- This holds even if  $J$  has no mappings and no axioms in the schemas

**Proof:** Let  $J = \langle \emptyset, \emptyset, \emptyset \rangle$ . Then,  $\text{sem}^D(J)$  contains every interpretation  $I$  for the alphabet of  $G$ . Therefore,  $\text{cert}(q, J, D) \neq \emptyset$  if and only if  $q$  is defined by a *tautological* FOL formula. Undecidability follows from ***undecidability of FOL Tautology*** (checking whether a FOL formula is a tautology)

We will study well behaved languages in the following lectures.

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Mappings

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. **Mapping between Global Schema e Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. **Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# **Information Integration Systems (recap)**

# Formalization of Information Integration Systems

- The architecture of an Information Integration Systems (I.I.S.) can be defined using **First-Order Logic**
  - However, we need to make some simplifying assumptions
- Data sources are represented as one single relational database
  1. No syntactic heterogeneity between sources
  2. A single logical theory (**source schema**) describes valid data sources
  3. Source schemas may contain constraints (FOL sentences)
- The result of the integration is a relational database.
  1. No syntactic heterogeneity between sources and the result of integration
  2. A single logical theory (**global schema**) describes the result of the integration
  3. Global schemas may contain constraints (FOL sentences)

# Information Integration Specification - Syntax

An information integration specification is a triple  $J = \langle G, M, S \rangle$ , where:

- **G** is the **global schema**
  - A relational database schema that describes how data resulting from the integration process should look like
  - A database **B** for **G** will be called **global database**, i.e., we assume that  $B \models G$
- **S** is the **source schema**
  - A relational database schema describing the data sources such that  $G \cap S = \emptyset$
  - A database **D** for **S** will be called **source database**, i.e., we assume that  $D \models S$
- **M** is the **mapping** between **source schema S** and **global schema G**
  - A set of mapping assertions  $m = \langle q_S, q_G \rangle$ , where  $q_S$  is a FOL query over **S** and  $q_G$  is a FOL query over **G**.  $q_S$  and  $q_G$  have the same arity!

# Information Integration System - Semantics

An information integration system is a pair  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$

A global database  $B$  **satisfies** the mapping  $M$  with respect to  $D$ , denoted by  $(B, D) \models M$ , if  $q_S(D) \subseteq q_G(B)$  for each  $m \in M$  (**sound source assumption**)

The semantics of the information integration system  $\langle J, D \rangle$  is the set of **global databases** that satisfy the **mapping**  $M$  w.r.t. to the **source database**  $D$ .

Formally:

$$\text{sem}(J, D) = \{ B \mid B \text{ is a } \text{global database} \text{ s.t. } (B, D) \models M \}$$

This semantics is based on **sound mappings**, but different semantics for mappings are possible.

# Querying an Information Integration System

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  and  $D$  is a **source database** for  $S$ , and query  $q$  over  $G$

The **certain answers** for  $q$  w.r.t.  $\langle J, D \rangle$  are those tuples of constants that occur in the answers for  $q$  over **every possible global database** in the semantics of  $\langle J, D \rangle$ .

Formally:

$$\text{cert}(q, J, D) = \{(c_1, \dots, c_n) \mid (c_1, \dots, c_n) \in q(B) \text{ for each } B \in \text{sem}(J, D)\}$$

That is, the *intersection* of the answers for  $q$  over all the **global database** in  $\text{sem}(J, D)$

# Different Forms of Mappings

We introduced the so called **sound mapping** semantics

- Why do we like it?
- Are there any other possible semantics?

What kind of queries can be used to specify mappings?

- Is there any limitation?

# **Semantics for Mappings**

# Sound Mappings

The **sound-mapping semantics**: Information at the **sources** may be partial but it is never wrong (i.e., it is valid information for the **global database**)

**EX:** The sources contain tables **Customers** and **Purchases**. The names of customers there represent actual people BUT there may be more people than those in the **sources**.

# Sound Mappings and Logic

Assume a information integration system  $\langle J, D \rangle$ , s.t.  $J = \langle G, M, S \rangle$  is and  $D$  is a **source database** for  $S$

The sound-mapping semantics of  $\langle J, D \rangle$  are defined as follows:

$$\text{sem}(J, D) = \{ B \mid q_S(D) \subseteq q_G(B) \text{ for each } m \in M \}$$

A mapping assertions  $m$  is of the form  $\langle q_S, q_G \rangle$ , where  $q_S = \{ \bar{x} \mid \varphi_S(\bar{x}) \}$  is a FOL query over  $S$  and  $q_G = \{ \bar{x} \mid \varphi_G(\bar{x}) \}$  is a FOL query over  $G$ .

Such mappings have a natural logical formulation:

$$\forall \bar{x}. \varphi_S(\bar{x}) \rightarrow \varphi_G(\bar{x})$$

# Sound Mappings: Example – the source DB

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

# Sound Mappings: Example – the Global Schema

Person

| Id | Name |
|----|------|
|----|------|

# Sound Mappings: Example – the IIS

Assume  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  with:

- $\mathbf{G} = \{Person_{/2}\}$
- $\mathbf{S} = \{Customers_{/3}\}$
- $\mathbf{M} = \{m\}$  where:
  - $m = \langle \{(n) | \exists d. \exists r. Customers(n, d, r)\}, \{(n) | \exists i. Person(i, n)\} \rangle$

and its logical formalization  $\rho$  defined as follows:

- $\rho : \forall n. \exists d. \exists r. Customers(n, d, r) \rightarrow \exists i. Person(i, n)$

Assume a **source database D** as defined previously.

We have that a **global database B** satisfies  $\mathbf{M}$  w.r.t.  $\mathbf{D}$ , written  $(\mathbf{B}, \mathbf{D}) \vDash \mathbf{M}$ , if  $q_S(\mathbf{D}) \subseteq q_G(\mathbf{B})$ .

We can prove that  $q_S(\mathbf{D}) \subseteq q_G(\mathbf{B})$  if and only if  $(\mathbf{B}, \mathbf{D}) \vDash \rho$

# Sound Mappings and Logic: Example

$$\forall n. \exists d. \exists r. \text{Customers}(n, d, r) \rightarrow \exists i. \text{Person}(i, n)$$

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |

OK

# Sound Mappings and Logic: Example

$$\forall n. \exists d. \exists r. \text{Customers}(n, d, r) \rightarrow \exists i. \text{Person}(i, n)$$

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |
| 3  | Kelly |

OK

# Sound Mappings and Logic: Example

$$\forall n. \exists d. \exists r. \text{Customers}(n, d, r) \rightarrow \exists i. \text{Person}(i, n)$$

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id        | Name      |
|-----------|-----------|
| Something | Amber     |
| Something | Rose      |
| Something | Something |

OK

# Sound Mappings and Logic: Example

$$\forall n. \exists d. \exists r. \text{Customers}(n, d, r) \rightarrow \exists i. \text{Person}(i, n)$$

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id | Name       |
|----|------------|
| 1  | Amber      |
| 2  | <b>Bob</b> |

**NO: Rose is missing!**

# Fully Logical Formalization of Information Integration Specifications - Syntax

An information integration specification is a triple  $J = \langle G, M, S \rangle$ , where:

- $G$  is the **global schema**: a logical theory over a relational alphabet  $A_G$
- $S$  is the **source schema**: a logical theory over a relational alphabet  $A_S$
- $M$  is the **mapping** between  $S$  and  $G$ : a logical theory over the relational alphabet  $A_S \cup A_G$ .

# Fully Logical Formalization of Information Integration Specifications - Semantics

Assume a information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$

The semantics of  $\langle J, D \rangle$  is:

$$\text{sem}(J, D) = \{ B \mid B \text{ is a global database s.t. } (B, D) \vDash M \}$$

Where  $(B, D) \vDash M$  means that  $(B, D) \vDash \rho_m$  for each  $m \in M$ , where  $\rho_m$  is the logical formalization of  $m$ .

Observe that in this formalization we are not forced to use sound mappings!

- **What kind of mappings are reasonable?**

# Fully Logical Formalization of Information Integration Specifications - Queries

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$ , and a query  $q$  over  $G$

The **certain answers** for  $q$  w.r.t.  $\langle J, D \rangle$  consist in the set:

$$\text{cert}(q, J, D) = \{(c_1, \dots, c_n) \mid (c_1, \dots, c_n) \in q(B) \text{ for each } B \in \text{sem}(J, D)\}$$

Since  $\text{sem}(J, D)$  is well defined also in the case of a fully logical formalization, we can use the **standard notion of certain answers**.

# Other Possible Semantics for Mappings

**The sound-mapping semantics.** Information at the sources may be partial but it is never wrong.

What are the fundamental assumptions?

- Possibly partial information
- Always correct information

Other options for  $\langle q_S, q_G \rangle$ ?

- Information may be *incorrect*, i.e.,  $q_S$  describes more than the answers for  $q_G$
- Information is exact, i.e.,  $q_S$  describes all the possible answers for  $q_G$

# Sound, Complete, and Exact Mappings

Let  $\langle J, D \rangle$  be an information integration system, where  $J = \langle G, M, S \rangle$  and  $D$  is a **source database** for  $S$

**The sound-mapping semantics:** sources contain partial but never wrong information regarding the global schema

$$\text{sem}(J, D) = \{ B \mid q_S(D) \subseteq q_G(B) \text{ for each mapping } \langle q_S, q_G \rangle \in M \}$$

**The complete-mapping semantics:** sources contain complete but possibly wrong information

$$\text{sem}(J, D) = \{ B \mid q_S(D) \supseteq q_G(B) \text{ for each mapping } \langle q_S, q_G \rangle \in M \}$$

**The exact-mapping semantics:** sources contain exact information regarding the global schema

$$\text{sem}(J, D) = \{ B \mid q_S(D) \equiv q_G(B) \text{ for each mapping } \langle q_S, q_G \rangle \in M \}$$

# Other Semantics for Mappings: Example

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

# Other Semantics for Mappings: Example

Assume  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  with:

- $\mathbf{G} = \{Person_{/2}\}$
- $\mathbf{S} = \{Customers_{/3}\}$
- $\mathbf{M} = \{m = \langle q_S, q_G \rangle\}$  where:
  - $q_S = \{ (n) \mid \exists d. \exists r. Customers(n, d, r)\}$
  - $q_G = \{ (n) \mid \exists i. Person(i, n)\}$

We examine the different semantics of for  $m$

# Other Semantics for Mappings: Example

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |

This is a model under all semantics

# Other Semantics for Mappings: Example

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |
| 3  | Kelly |

This is a model only under the sound semantics

# Other Semantics for Mappings: Example

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Rose  |
| 3  | Rose  |

This is a model under all semantics

# Other Semantics for Mappings: Example

Customers

| Name  | DOB        | Residency      |
|-------|------------|----------------|
| Amber | 10/20/2030 | New Amsterdam  |
| Rose  | 10/20/1930 | Constantinople |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |

This is a model only under complete semantics

# Logical Formalization of the Different Semantics

Assume a mapping assertion  $m = \langle q_S, q_G \rangle$  where  $q_S$  and  $q_G$  are expressed as FOL queries:

$$q_S = \{ \bar{x} \mid \varphi_S(\bar{x}) \} \text{ and } q_G = \{ \bar{x} \mid \varphi_G(\bar{x}) \};$$

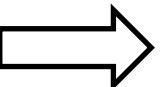
## Sound Semantics of $m$

- $\forall \bar{x}. \varphi_S(\bar{x}) \rightarrow \varphi_G(\bar{x})$

## Complete Semantics of $m$

- $\forall \bar{x}. \varphi_S(\bar{x}) \leftarrow \varphi_G(\bar{x})$

## Exact Semantics of $m$

- $\forall \bar{x}. \varphi_S(\bar{x}) \rightarrow \varphi_G(\bar{x})$
  - $\forall \bar{x}. \varphi_S(\bar{x}) \leftarrow \varphi_G(\bar{x})$
- 
- $$\forall \bar{x}. \varphi_S(\bar{x}) \leftrightarrow \varphi_G(\bar{x})$$

# Exact Mappings: Some Counterintuitive Behavior

- One could be tempted to think that **exact** mappings are the most reasonable semantics for Information Integration Systems.
- Actually, **complete** and **exact** mappings can have a very **counterintuitive behavior**
  - Information at the **sources** is influenced by information in the target schema!
  - Simple assertions may give rise to **inconsistent systems (i.e., systems with no semantics)**

# Exact Mappings: Example

Let  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  be such that

- $\mathbf{G} = \{Person_{/2}\}$
- $\mathbf{S} = \{Customers_{/3}, Purchases_{/3}\}$
- $\mathbf{M} = \{m_1, m_2\}$  where:
  - $m_1 : \forall n. \exists d. \exists r. Customers(n, d, r) \leftrightarrow \exists i. Person(i, n)$
  - $m_2 : \forall n. \exists i. \exists p. Purchases(i, n, p) \leftrightarrow \exists i. Person(i, n)$

Let  $\mathbf{D}$  be a **source database** as follows:

Customers

| Name  | DOB        | Residency     |
|-------|------------|---------------|
| Amber | 10/20/2030 | New Amsterdam |

Purchases

| ID   | Customer Name | Product |
|------|---------------|---------|
| O002 | Dylan         | P001    |

We can show that that  $\text{sem}(\mathbf{J}, \mathbf{D}) = \emptyset$

# Exact Mappings: Example

Customers

| Name  | DOB        | Residency     |
|-------|------------|---------------|
| Amber | 10/20/2030 | New Amsterdam |

Person

| Id | Name |
|----|------|
|    |      |

Purchases

| ID   | Customer Name | Product |
|------|---------------|---------|
| O002 | Dylan         | P001    |

$$m_1 : \forall n. \exists d. \exists r. Customers(n, d, r) \leftrightarrow \exists i. Person(i, n)$$
$$m_2 : \forall n. \exists i. \exists p. Purchases(i, n, p) \leftrightarrow \exists i. Person(i, n)$$

# Exact Mappings: Example

Customers

| Name  | DOB        | Residency     |
|-------|------------|---------------|
| Amber | 10/20/2030 | New Amsterdam |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |

Purchases

| ID   | Customer Name | Product |
|------|---------------|---------|
| O002 | Dylan         | P001    |

$m_1 : \forall n. \exists d. \exists r. Customers(n, d, r) \leftrightarrow \exists i. Person(i, n)$

$m_2 : \forall n. \exists i. \exists p. Purchases(i, n, p) \leftrightarrow \exists i. Person(i, n)$

Violate  $m_2$

# Exact Mappings: Example

Customers

| Name  | DOB        | Residency     |
|-------|------------|---------------|
| Amber | 10/20/2030 | New Amsterdam |

Person

| Id | Name  |
|----|-------|
| 2  | Dylan |

Purchases

| ID   | Customer Name | Product |
|------|---------------|---------|
| O002 | Dylan         | P001    |

$m_1 : \forall n. \exists d. \exists r. Customers(n, d, r) \leftrightarrow \exists i. Person(i, n)$

$m_2 : \forall n. \exists i. \exists p. Purchases(i, n, p) \leftrightarrow \exists i. Person(i, n)$

Violate  $m_1$

# Exact Mappings: Example

Customers

| Name  | DOB        | Residency     |
|-------|------------|---------------|
| Amber | 10/20/2030 | New Amsterdam |

Person

| Id | Name  |
|----|-------|
| 1  | Amber |
| 2  | Dylan |

Purchases

| ID   | Customer Name | Product |
|------|---------------|---------|
| O002 | Dylan         | P001    |

$m_1 : \forall n. \exists d. \exists r. Customers(n, d, r) \leftrightarrow \exists i. Person(i, n)$   
 $m_2 : \forall n. \exists i. \exists p. Purchases(i, n, p) \leftrightarrow \exists i. Person(i, n)$

Violate both  $m_1$  and  $m_2$

# Query Answering Under Sound Mappings

From now on we focus only on **sound mappings** and assume their logical formalization.

We have a problem:

**Thm:** Given a database **D**, an information integration system  $\langle J, D \rangle$ , and a FOL query **q**, computing  $\text{cert}(q, J, D)$  is **undecidable** (actually to be undecidable is the associated *decision problem*)

The theorem above tells us that we cannot expect to answer queries over and Information integration system with **arbitrarily expressive mappings**.

**We need to impose further restrictions on the languages involved**

# Complexity of Query Answering

**Thm:** Given a database  $D$ , an information integration system  $\langle J, D \rangle$ , and a FOL query  $q$ , computing  $\text{cert}(q, J, D)$  is **undecidable** (actually to be undecidable is the associated *decision problem* – see later).

**Proof:** The undecidability follows from the undecidability of the validity problem in FOL.

Assume to have:

- a mapping  $M = \{\forall x. T(x) \rightarrow \psi(x)\}$  with  $\psi = (\varphi \rightarrow R(b))$ .
- a query  $q = \exists x. R(x)$
- a source database  $D$  such that  $D(T) = \{a\}$

We have that:

- If  $\varphi$  is a **tautology** then  $\langle b \rangle \in B(R)$  for each  $B \in \text{sem}(J, D)$ .
  - We conclude  $\text{cert}(q, J, D) \neq \emptyset$
- If  $\varphi$  is not a **tautology** then  $B(R) = \emptyset$  for some  $B \in \text{sem}(J, D)$ .
  - We conclude  $\text{cert}(q, J, D) = \emptyset$

# Restricting the Language

We need to impose further restrictions on the languages involved.

The restriction that we will adopt (see the slides on query answering) is to assume mappings that only use **conjunctive queries**

$$\forall \bar{x} \ \varphi(\bar{x}) \rightarrow \psi(\bar{x})$$

where both  $\varphi$  and  $\psi$  are **existential conjunctions**, i.e., formulae of the form

$$\exists y_1, \dots, \exists y_n P_1(\bar{z}_1) \wedge \dots \wedge P_k(\bar{z}_k)$$

# **Different Forms of Mappings: GAV, LAV, and GLAV**

# Different Forms of Mappings

Inside **sound conjunctive mappings**, we distinguish three important and widely used classes:

The **global schema** is defined in terms of the data at the **sources**

- **GAV: Global-As-View**

**Data at sources** is defined in terms of the **global schema**

- **LAV: Local-As-View**

A mixed and more powerful approach

- **GLAV: Global-Local-As-View**

# Global-As-View

A GAV mapping describes the **global schema  $\mathbf{G}$**  in terms of the **source schema  $\mathbf{S}$**

A GAV **mapping assertion** is a pair  $\langle q_s, g \rangle$  where  $q_s$  is a FOL query over the **source schema** and  $g \in \mathbf{G}$  is a **single atom** of the **global schema**.

Logical formalization of GAV **mapping assertions**

$$\forall \bar{x} . \varphi_s(\bar{x}) \rightarrow g(\bar{x})$$

A **GAV mapping  $\mathbf{M}$**  is a finite set of GAV mapping assertions

# Global As View – Observations

- GAV mappings provide direct information on what data populates the relations of the **global schema**.
- Elements of the **global schema** are considered **views** over the data sources (**Global-As-View**).
- No direct method to cope with incompleteness at the sources.
  - No quantification in the right-hand side of mapping assertions
- GAV is a very natural and simple way to define a mapping.
  - Pretty much the same as defining views over a relational database.

# Example of GAV Mapping

- **Global Schema G** containing:
  - **Movie**(Title, Year, Director)
  - **European**(Director)
  - **Review**(Title, Critique)
- **Source Schema S** containing:
  - **r1**(Title, Year, Director): Movies since 1960 with European directors
  - **r2**(Title, Critique): Reviews of movies since 1990

**GAV Mapping M={m<sub>1</sub>,m<sub>2</sub>,m<sub>3</sub>}**:

- **m<sub>1</sub>**:  $\forall t, y, d. r_1(t, y, d) \rightarrow Movie(t, y, d)$
- **m<sub>2</sub>**:  $\forall d. \exists y, t. r_1(t, y, d) \rightarrow European(d)$
- **m<sub>3</sub>**:  $\forall t. \exists r. r_2(t, r) \rightarrow Review(t, r)$

## Local As View

LAV mappings describe the **source schema** in terms of the **global schema**

A LAV **mapping assertion** is a pair  $\langle s, q_G \rangle$  where  $q_G$  is a FOL query over the **global schema** and  $s \in S$  is a **single atom** of the **source schema**.

Logical formalization of LAV mapping assertions

$$\forall \bar{x} \ s(\bar{x}) \rightarrow \varphi_G(\bar{x})$$

A LAV **mapping** is a **finite** set of LAV mapping assertions.

- We assume that every element of the source schema is **defined exactly once**

## Local As View – Observations

- LAV mappings **do not provide** direct information on what data populates the **global schema**.
  - Single elements of the **global schema** may not be explicitly defined
- **Mappings** are defined considering **data sources** as **views** over the **global schema (Local-As-View)**.
- They provide a direct method to cope with **incompleteness** at the **sources**.
  - Quantification can appear in the right-hand side of mapping assertions
- LAV is a sophisticated way to define a mapping: modelling requires attention

# Example of LAV Mapping

- **Global Schema G** containing:
  - **Movie**(Title, Year, Director)
  - **European**(Director)
  - **Review**(Title, Critique)
- **Source Schema S** containing:
  - **r1**(Title, Year, Director): Movies since 1960 with European directors
  - **r2**(Title, Critique): Reviews of movies since 1990

**LAV Mapping M={m<sub>1</sub>,m<sub>2</sub>}**:

- $m_1: \forall t, y, d. r_1(t, y, d) \rightarrow \text{Movie}(t, y, d) \wedge \text{European}(d)$
- $m_2: \forall t, r. r_2(t, r) \rightarrow \exists y, d. \text{Review}(t, r) \wedge \text{Movie}(t, y, d)$

# GAV vs LAV Mapping

- **GAV**
  - Quality depends on how well we can compile the **sources** into the **global schema**
  - Changing on the **sources** may reflect heavily on the mapping
  - Query processing is very easy and based on some sort of *unfolding* (see later)
- **LAV**
  - Quality depends on how well we can characterize the **sources** using the **global schema**
  - Highly modularity and extensibility. In case of changes at the **sources** we can simply redefine the portion that has changed.
  - Query processing is not straightforward (see later)

# Global and Local As View

- GLAV can define the **source schema** using the **global schema** and vice versa (**Global-And-Local-As-View**) and define complex constructions
- A **GLAV mapping assertion** is a pair  $\langle q_s, q_G \rangle$  where  $q_G$  is a FOL query over the **global schema** and  $q_s$  is a FOL query over the **source schema**
- Logical formalization GLAV mapping assertions

$$\forall \bar{x} \varphi_S(\bar{x}) \rightarrow \varphi_G(\bar{x})$$

- A GLAV **mapping** is a **finite** set of GLAV mapping assertions.

# Global and Local As View – Observations

- GLAV mappings **do not provide** direct information on what data populates the **global schema**.
  - Single elements of the **global schema** may not be explicitly defined
- Direct method to cope with incompleteness at the **sources**.
  - Quantification can appear in both sides of the assertions
- GLAV is a sophisticated way to define a mapping.
  - Modelling requires attention.

# Example of GLAV Mapping

- **Global Schema G** containing:
  - Work(Person, Project)
  - Area(Project, Field)
- **Source Schema S** containing:
  - HasJob(Person, Field)
  - Teaches(Professor, Course)
  - In(Course, Field)
  - Get(Researcher, Grant)
  - For(Grant, Project)

**GLAV Mapping M={m<sub>1</sub>,m<sub>2</sub>,m<sub>3</sub>}**:

- $m_1 = \forall r, f. \text{HasJob}(r, f) \rightarrow \exists p. \text{Work}(r, p) \wedge \text{Area}(p, f)$
- $m_2 = \forall r, f. \exists c. \text{Teaches}(r, c) \wedge \text{In}(c, f) \rightarrow \exists p. \text{Work}(r, p) \wedge \text{Area}(p, f)$
- $m_3 = \forall r, p. \exists c. \text{Get}(r, g) \wedge \text{For}(g, p) \rightarrow \text{Work}(r, p)$

## Some Technical Consideration

We can think of GLAV mappings as the composition of a LAV and a GAV mapping. Indeed, each GLAV mapping  $\langle q_S, q_G \rangle$  can be rewritten as

$$\langle q_S, p \rangle \cup \langle p, q_G \rangle$$

where  $p$  is a fresh predicate of the same arity as the two queries  $q_S$  and  $q_G$

**Example:**

$$m_2 = \langle \{(r,f) \mid \exists c. \text{Teaches}(r,c) \wedge \text{In}(c,f)\}, \{(r,f) \mid \exists p. \text{Work}(r,p) \wedge \text{Area}(p,f)\} \rangle$$

- $\langle \{(r,f) \mid \exists c. \text{Teaches}(r,c) \wedge \text{In}(c,f)\}, \{(r,f) \mid p(r,f)\} \rangle$
- $\langle \{(r,f) \mid p(r,f)\}, \{(r,f) \mid \exists p. \text{Work}(r,p) \wedge \text{Area}(p,f)\} \rangle$

From an algorithmic perspective, we can consider LAV and GLAV the **same**.

# Exercise

## **Exercise – Scenario**

# Description of the Scenario

You are asked to design an information integration system for a company.

**Data sources** consist of three tables that do **not** contain NULL values:

- **VTables** (id, plateNumber, type, ownerID, ownerType)  
This table stores information about vehicles.
  - The attributes **plateNumber** and **ownerID** can contain the value '**000**' meaning that the information is missing
  - The attribute **type** can be '*motorcycle*', '*car*', or '*truck*'
  - The attribute **ownertype** can be either '*person*' or '*company*'
- **PTable**(code, age, name, address, phone)  
This table stores information about persons
- **CTable**(code, name, address, phone)  
This table stores information about companies
- **Constraint:** **VTable(ownerID)** is either a **Person(code)** or a **Company(code)** or **000**.
  - $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \rightarrow \exists x, y, z, w. PTable(o, x, y, z, w) \vee CTable(o, x, y, z) \vee (o = 000)$

# Description of the Scenario

You are asked to design an information integration system for a company.

- **Global Schema alphabet** is defined as follows:
  - **Vehicles<sub>/1</sub>**: set of vehicles.
  - **Motorcycle<sub>/1</sub>**: set of motorcycles
  - **Car<sub>/1</sub>**: set of cars.
  - **Owns<sub>/2</sub>**: Owns(x, y) means x is the owner of y.
  - **hasPlate<sub>/2</sub>**: hasPlate(x, y) means that vehicle x has plate number y

# Requirements.

1. Define suitable axioms for the [global schema](#)
2. Define a suitable GAV mapping.
3. Define a suitable GLAV mapping.
4. Extend the global schema to accommodate the information in the sources.

# **Solution**

# Solution: Axioms

1. Define suitable axioms for the global schema

- $\forall x. \text{Motorcycle}(x) \rightarrow \text{Vehicle}(x)$ : Motorcycles are Vehicles
- $\forall x. \text{Car}(x) \rightarrow \text{Vehicle}(x)$ : Cars are Vehicles
- $\forall x. \text{Motorcycle}(x) \rightarrow \neg \text{Car}(x)$ : Motorcycles are not Cars and vice versa
- $\forall x. \text{Vehicle}(x) \rightarrow \exists y. \text{hasPlate}(x, y)$ : Every Vehicle has a license plate
- $\forall x. \exists y. \text{hasPlate}(x, y) \rightarrow \text{Vehicle}(x)$ : What has a license plate is a vehicle
- $\forall x. \text{Vehicle}(x) \rightarrow \exists y. \text{Owns}(y, x)$ : Every Vehicle has a owner
- $\forall x. \forall y. \forall y'. \text{hasPlate}(x, y) \wedge \text{hasPlate}(x, y') \rightarrow (y = y')$ : the predicate hasPlate is functional

## Solution: GAV Mapping

2. Define a suitable GAV mapping.

- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \rightarrow Vehicle(i)$
- $\forall i, p, o, t_o. VTable(i, p, Motorcycle, o, t_o) \rightarrow Motorcycle(i)$
- $\forall i, p, o, t_o. VTable(i, p, Car, o, t_o) \rightarrow Car(i)$
- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \wedge (o \neq 000) \rightarrow Owns(o, i)$
- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \wedge (p \neq 000) \rightarrow hasPlate(o, i)$

# GLAV Mapping

## 3. Define a suitable GAV mapping

- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \rightarrow Vehicle(i)$
- $\forall i, p, o, t_o. VTable(i, p, Motorcycle, o, t_o) \rightarrow Motorcycle(i)$
- $\forall i, p, o, t_o. VTable(i, p, Car, o, t_o) \rightarrow Car(i)$
- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \wedge (o = 000) \rightarrow \exists x. Owns(x, i)$
- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \wedge (p = 000) \rightarrow \exists x. hasPlate(i, x)$
- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \wedge (o \neq 000) \rightarrow Owns(o, i)$
- $\forall i, p, t_v, o, t_o. VTable(i, p, t_v, o, t_o) \wedge (p \neq 000) \rightarrow hasPlate(o, i)$

# Solution: Global Schema

4. Extend the global schema to accommodate all the information in the sources.

- $\text{Vehicles}_{/1}$ ,  $\text{Motorcycle}_{/1}$ ,  $\text{Car}_{/1}$ : Set of cars.
- $\text{Owns}_{/2}$ ,  $\text{hasPlate}_{/2}$
- $\text{Person}_{/1}$ : set of persons
- $\text{Company}_{/1}$ : set of companies
- $\text{hasCode}_{/2}$ :  $\text{hasCode}(x, y) \times \text{has code } y$
- $\text{hasAddress}_{/2}$ :  $\text{hasAddress}(x, y) \times \text{has address } y$
- $\text{hasPhNum}_{/2}$ :  $\text{hasPhNum}(x, y) \times \text{has phone number } y$

## Solution: Axioms

4. Extend the global schema to accommodate all the information in the sources.

- $\forall x. \text{Motorcycle}(x) \rightarrow \text{Vehicle}(x)$
- $\forall x. \text{Motorcycle}(x) \rightarrow \text{Car}(x)$
- $\forall x. \text{Vehicle}(x) \rightarrow \exists y. \text{hasPlate}(x, y)$
- $\forall x. \exists y. \text{hasPlate}(x, y) \rightarrow \text{Vehicle}(x)$
- $\forall x. \text{Vehicle}(x) \rightarrow \exists y. \text{Owns}(y, x)$
- $\forall x. \forall y. \forall y'. \text{hasPlate}(x, y) \wedge \text{hasPlate}(x, y') \rightarrow (y = y')$
- $\forall x. \text{Person}(x) \rightarrow \exists y. \text{hasCode}(x, y)$
- $\forall x. \text{Company}(x) \rightarrow \exists y. \text{hasCode}(x, y)$
- ...

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Incomplete Information

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. **Incomplete information databases**
8. Query answering over IISs
9. Ontology-Based Data Management (OBDM)
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# Query Answering Over Databases (recap)

# Databases as an Interpretation

We can see a database instance  $\mathbf{D}$  as a (finite) interpretation  $I$ .

- Our **domain** is the set  $\mathbf{dom}(\mathbf{D})$  of attribute values
- Our **alphabet** is formed by:
  - the set **Rels** of relational predicates  $R_{/n}$
  - the set **Cons** of constants (we have a function  $c$  with arity 0 (constant) for each element  $c$  in  $\mathbf{dom}$ ).

We use **databases as interpretations**. Hence, given a database  $\mathbf{D}$  over a schema  $S = \{R_1, \dots, R_n\}$ , we implicitly see  $\mathbf{D}$  as the interpretation  
 $I = (\mathbf{dom}(\mathbf{D}), \cdot^I)$ , where:

- $R^I = D(R)$  for each  $R \in S$  and  $R^I = \emptyset$  for each  $R \notin S$
- $c^I = c$  for each  $c \in \mathbf{Cons}$  (each constant is interpreted into itself)

# Answers for Complete Database

Consider to have instance  $\mathbf{D}$  of a schema  $\mathbf{S}$  over a domain  $dom$ , we can see  $\mathbf{D}$  as a FOL **interpretation**.

So, given a FOL formula  $\varphi$  and an assignment  $\alpha$ , we know if  $\mathbf{D}, \alpha \models \varphi$

The **answers** for a query  $q = \{(x_1, \dots, x_n) \mid \varphi(x_1, \dots, x_n)\}$  over a database  $\mathbf{D}$  are the set  $q(\mathbf{D})$  defined as follows:

$$q(\mathbf{D}) = \{< c_1, \dots, c_n > \in D^n \mid \mathbf{D}, < c_1, \dots, c_n > \models \varphi(x_1, \dots, x_n)\}$$

# Example

Consider the following database  $\mathbf{D}$  and the FOL query:

$$q = \{(x) \mid \exists y. \exists z. (Customers(y, x) \wedge Orders(z, y, P002))\}$$

**Products**

| Code | Color |
|------|-------|
| P001 | Red   |
| P002 | Green |
| P003 | Blue  |

**Customers**

| Code | Name  |
|------|-------|
| C001 | Eryn  |
| C002 | Dylan |
| C003 | Cora  |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | P001    |
| O002 | C002     | P002    |
| O003 | C001     | P001    |

We have that:

$$q(\mathbf{D}) = \langle Dylan \rangle$$

# Incomplete Data

# More than Completeness

The general notion of sound mappings gives rise to **incomplete information**

Consider for instance the following (G)LAV mapping assertion:

$$m: \forall t, r. \textcolor{brown}{r_2(t, r)} \rightarrow \exists y, d. \textcolor{blue}{Review}(t, r) \wedge \textcolor{blue}{Movie}(t, y, d)$$

Given a source database **D**, in general there are several solutions for a set of (G)LAV assertions (i.e., different databases that are legal w.r.t. **G** and that satisfy **M** w.r.t. **D**)

→ **Mapping may generate incompleteness**

Especially for *materialization-based techniques*, we need a mechanism to capture incompleteness in databases.

So far, we considered databases with **complete information**: a tuple is either in a relation or it is not

**We need a new notion of databases that allows incompleteness**

# Complete Databases (1/2)

So far we always assumed that databases are complete.

In what sense?

**Products**

| Code | Color |
|------|-------|
| P001 | Red   |
| P002 | Green |
| P003 | Blue  |

**Customers**

| Code | Name  |
|------|-------|
| C001 | Eryn  |
| C002 | Dylan |
| C003 | Cora  |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | P001    |
| O002 | C002     | P002    |
| O003 | C001     | P001    |

## Complete Databases (2/2)

We know **everything** about the objects represented in the database.

- We know the color of all products, the names of all customers...

We know **ALL** the objects of interest

- We know all products, all customers...

We also know what an object is **NOT**

- For example, we know that P001 is NOT a customer code.

# Queries Over Complete Databases

What are the answers for the following queries?

- Return code and color of all the products.
  - $\{(x, y) \mid Products(x, y)\}$
  - **Answers:** {<P001,Red>, <P002,Green>, <P003,Blue>}
- Return the codes of products that are not Red.
  - $\{(x) \mid \exists y. \neg Products(x, Red) \wedge Products(x, y)\}$
  - **Answers:** {<P002>, <P003>}

**Products**

| Code | Color |
|------|-------|
| P001 | Red   |
| P002 | Green |
| P003 | Blue  |

# Incomplete Data

Data may be incomplete in two different ways:

- Some **value** may be missing.
  - E.g., we may **not** know the color of P003
  - Return code and color of all the products.
    - $\{(x, y) \mid \text{Products}(x, y)\}$
    - **Answers:**  $\{(P001, \text{Red}), (P002, \text{Green}), (P003, ??)\}$
- Some **tuple** may be missing
  - E.g., there may be more products than those in the table Products.
  - Return the codes of products that are not Red.
    - $\{(x) \mid \forall y. \neg \text{Products}(x, \text{Red})\}$
    - **Answers:**  $\{P002, P003, \dots\}$  what else?

# Representing Incomplete Data

This scenario poses us two main challenges:

- 1. How do we represent the missing information?**
- 2. What does it mean to answer a query over such database?**

# **Incomplete Information in the Relational Model**

# Incomplete Databases: Real-World Solutions

Often “normal” values of the domain (such as 0, the empty string, “9999”, etc) are used to represent **missing information**.

However, this is a **mistake** for a host of reasons:

- There might not be any “unused values”.
- Values that are “unused” up to a certain point, might become used later.
- The token used for the missing value carries no meaning. Different programmers\users may use different tokens.

**How do we model (correctly) incomplete information in the relational model?**

# Incomplete Information in the Relational Model

In the relational model, we use a **specific value** to represent incomplete information: the **null** value so defined:

- Let **null**  $\notin \text{dom}$  be a new symbol.
- We allow the use of **null** in a database instance.
  - Tuples can now use **null** together with values from **dom**.
  - In other words,  $t[A]$ , for every attribute  $A$ , is either a value in **dom** or is **null**.
- We often refer to this model of nulls as **Codd Nulls**.

# A Database with Codd Nulls

Some values are now missing

**Products**

| Code | Color |
|------|-------|
| P001 | Red   |
| P002 | Green |
| P003 | Null  |

**Customers**

| Code | Name |
|------|------|
| C001 | Eryn |
| C002 | Null |
| C003 | Cora |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | P001    |
| O002 | C002     | P002    |
| O003 | C001     | P001    |

# A Database with Codd Nulls – another example

**Example:** In our domain, county towns have government offices, other cities do not.

We know that:

- **Rome** is a county town; so it has a government office which is in **Via IV Novembre**.
- **Florence** is a county town; so it has a government office, but we **do not know** its address
- **Tivoli** is not a county town; so **it has no** government office
- **Prato** has recently become a county town; has the government office been established?  
**We don't know**

| City     | GovAddress      |
|----------|-----------------|
| Roma     | Via IV Novembre |
| Florence | Null            |
| Tivoli   | Null            |
| Prato    | Null            |

# Interpretations of the Null Value

We have (at least) three different interpretations of the null value:

- **unknown value**: there is a domain value, but it is **not known** (we do not know the address of the government office in Florence)
- **non-existent value**: the attribute is **not applicable** for the tuple (Tivoli has no government office)
- **no-information value**: we **don't know** whether a value **exists or not** (Prato is a new county town, we do not know if the government already exists or not) - this is the disjunction (logical or) of the other two

The common assumption is that **NULL** represents an **unknown value**.

# Handling NULLs

How should we deal with null values?

- *null* = *null*?
- *null* ≠ *null*?
- Neither!

Comparisons involving NULLS are evaluated using a three-valued logic that allows for the **unknown** truth value:

- $(1 = 1)$  is **true**.
- $(1 = 2)$  is **false**.
- $(1 = \text{NULL})$  is **unknown**.

# Kleene's Three-Valued Logic

Special truth tables are used to handle the third truth value.  
This is named **Kleene's three-valued logic**.

| NOT(A) |          |
|--------|----------|
| A      | $\neg A$ |
| F      | T        |
| U      | U        |
| T      | F        |

| AND(A, B)    |   |
|--------------|---|
| A $\wedge$ B |   |
|              | B |
|              |   |
| A            |   |
| F            |   |
| U            |   |
| T            |   |
| B            |   |
| F            |   |
| U            |   |
| T            |   |

| A | $\neg A$ |
|---|----------|
| F | T        |
| U | U        |
| T | F        |

| OR(A, B)   |   |
|------------|---|
| A $\vee$ B |   |
|            | B |
|            |   |
| A          |   |
| F          |   |
| U          |   |
| T          |   |
| B          |   |
| F          |   |
| U          |   |
| T          |   |

| A | $\neg A$ |
|---|----------|
| F | F        |
| U | U        |
| T | T        |

# Where NULL is Used

- The use of NULL was first proposed by Codd in the 70s
  - In the first series of papers presenting the relational model
- SQL uses this notion of NULL!
  - SQL engines evaluate conditions using three truth values
  - Tuples evaluating to true are then returned.
- The following SQL query may not return the whole R. Why?
  - `Select * FROM R WHERE R.a = 1 OR R.a <> 1`

# **Marked Nulls**

# Incomplete Databases

- Having a single **NULL** is unsatisfactory in many ways.
  - We cannot say when two unknown values are the **same**
  - The use of three-valued logic is often counterintuitive.
- A more reasoned approach to incomplete information in relational databases is the use of ***multiple null values***.
  - We often refer to this model of nulls as **Marked Nulls**.
- More formally, we assume a countably infinite set named **nulls**.
  - We assume the set **nulls** disjoint from the set **dom** of database values.
  - We understand the symbols in **nulls** as ***null*** values.

# A Database with Marked Nulls

Values are missing but we know that **some are the same**.

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Customers**

| Code | Name              |
|------|-------------------|
| C001 | Eryn              |
| C002 | Null <sub>2</sub> |
| C003 | Cora              |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Evaluating Queries over incomplete databases: Intuition

We call a database instance **complete** if it **does not contain nulls**.

What does it mean to answer a query over a database?

The case of complete databases is clear!

We want to avoid the use of many-valued logics: counterintuitive behavior.

**Intuition:** in case of missing values, we want to return what is **certain**.

# Example

How would you answer the following queries?

- $\{(x, y) \mid Products(x, y)\}$
- $\{(x) \mid \exists y. Products(x, y)\}$
- $\{(x) \mid \exists y. Products(y, x)\}$
- $\{(z) \mid \exists x. \exists y. Products(x, Red) \wedge Orders(y, z, x)\}$

Products

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

Orders

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Models of a Database

To capture the notion of incomplete information, we use the notion of **valuation**.

A valuation for the set **nulls** is a function  $\nu : \text{nulls} \rightarrow \text{dom}$ .

- Given a database instance  $D$ , we use  $\nu(D)$  to denote the instance obtained by replacing each **null**  $n$  occurring in  $D$  with  $\nu(n)$
- Clearly,  $\nu(D)$  produces a **complete database**.

The set of **models of a database**  $D$  (denote by  $\text{Mod}(D)$ ) is the set:

$$\{\nu(D) \mid \nu \text{ is a } \text{valuation}\}$$

# Models: Example (1\4)

A database with **marked nulls**

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Customers**

| Code | Name              |
|------|-------------------|
| C001 | Eryn              |
| C002 | Null <sub>2</sub> |
| C003 | Cora              |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Models: Example (2\4)

Different nulls may go into **different** values

**Products**

| Code | Color |
|------|-------|
| a    | Red   |
| P002 | Green |
| P003 | b     |

**Customers**

| Code | Name |
|------|------|
| C001 | Eryn |
| C002 | c    |
| C003 | Cora |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | a       |
| O002 | C002     | P002    |
| O003 | C001     | a       |

# Models: Example (3\4)

Different nulls may go into the **same** value

**Products**

| Code | Color |
|------|-------|
| a    | Red   |
| P002 | Green |
| P003 | a     |

**Customers**

| Code | Name |
|------|------|
| C001 | Eryn |
| C002 | a    |
| C003 | Cora |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | a       |
| O002 | C002     | P002    |
| O003 | C001     | a       |

# Models: Example (4\4)

Nulls may go into values that were **already** in the database

**Products**

| Code | Color |
|------|-------|
| Red  | Red   |
| P002 | Green |
| P003 | C001  |

**Customers**

| Code | Name |
|------|------|
| C001 | Eryn |
| C002 | Eryn |
| C003 | Cora |

**Orders**

| Code | Customer | Product |
|------|----------|---------|
| O001 | C001     | Red     |
| O002 | C002     | P002    |
| O003 | C001     | Red     |

## Certain Answers

To capture the notion of certainty for query answers, we adopt the notion of **certain answers**.

Consider a FOL query  $\mathbf{q} = \{(x_1, \dots, x_n) \mid \varphi(x_1, \dots, x_n)\}$

A tuple  $\langle c_1, \dots, c_n \rangle \in \mathbf{dom}^n$  is a **certain answer** for  $\mathbf{q}$  over a database  $\mathbf{D}$ , if  $\langle c_1, \dots, c_n \rangle \in q(\nu(\mathbf{D}))$ , for **every valuation**  $\nu$ .

$cert(q, \mathbf{D})$  will denote the set of **certain answers** for  $\mathbf{q}$  over  $\mathbf{D}$ .

# Exercise 1

What are the **certain answers** for the following queries?

- $\{(x, y) \mid Products(x, y)\}$
- $\{(x) \mid \exists y. Products(x, y)\}$
- $\{(x) \mid \exists y. Products(y, x)\}$
- $\{(z) \mid \exists x. \exists y. Products(x, Red) \wedge Orders(y, z, x)\}$
- $\{(x) \mid \exists y. Products(x, y) \wedge \neg(y = Green)\}$
- $\{(x) \mid \exists y. \exists z. \exists w. Orders(x, y, z) \wedge Products(z, w)\}$

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| 0001 | C001     | Null <sub>3</sub> |
| 0002 | C002     | P002              |
| 0003 | C001     | Null <sub>3</sub> |

# Exercise 1: Solution (1\6)

What are the **certain answers** for the following queries?

- $\{(x, y) \mid Products(x, y)\}$ 
  - $\{(P002, \text{Green})\}$

---

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Exercise 1: Solution (2\6)

What are the **certain answers** for the following queries?

- $\{(x) \mid \exists y. Products(x, y)\}$ 
  - $\{(P002), (P003)\}$

---

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Exercise 1: Solution (3\6)

What are the **certain answers** for the following queries?

- $\{(x) \mid \exists y. Products(y, x)\}$ 
  - {Green, Red}

---

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Exercise 1: Solution (4\6)

What are the **certain answers** for the following queries?

- $\{(z) \mid \exists x. \exists y. Products(x, Red) \wedge Orders(y, z, x)\}$ 
  - {C001}

---

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| 0001 | C001     | Null <sub>3</sub> |
| 0002 | C002     | P002              |
| 0003 | C001     | Null <sub>3</sub> |

# Exercise 1: Solution (5\6)

What are the **certain answers** for the following queries?

- $\{(x) \mid \exists y. Products(x, y) \wedge \neg (x = Green)\}$ 
  - $\{\}$

---

**Products**

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

**Orders**

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Exercise 1: Solution (6\6)

What are the **certain answers** for the following queries?

- $\{(x) \mid \exists y. \exists z. \exists w. Orders(x, y, z) \wedge Products(z, w)\}$ 
  - {O001, O002, O003}

---

Products

| Code              | Color             |
|-------------------|-------------------|
| Null <sub>3</sub> | Red               |
| P002              | Green             |
| P003              | Null <sub>1</sub> |

Orders

| Code | Customer | Product           |
|------|----------|-------------------|
| O001 | C001     | Null <sub>3</sub> |
| O002 | C002     | P002              |
| O003 | C001     | Null <sub>3</sub> |

# Complexity of Certain Answers: Preliminaries

## What is the complexity of computing Certain Answers?

We have different kind of complexity:

- **Combined Complexity**: The complexity of the whole problem.
- **Data Complexity**: The complexity of the problem for a fixed query.
- **Query Complexity**: The complexity of the problem for a fixed interpretation.

# Complexity of Certain Answers

**Thm:** Let  $q$  be a FOL query. Checking whether  $\bar{a} \in \text{cert}(q, D)$  can be done in EXPTIME in **Combined Complexity** and is NP-Complete in **Data Complexity**.

# **Conjunctive Query Answering in CWA**

# Conjunctive Query

- Let  $\varphi(x_1, \dots, x_n)$  be a FOL formula with free variables  $x_1, \dots, x_n$
- $\varphi(x_1, \dots, x_n)$  is an **Existential Conjunction** if it is of the form

$$\exists y_1, \dots, \exists y_n P_1(\bar{z}_1) \wedge \dots \wedge P_k(\bar{z}_k)$$

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Conjunctive Query** if  $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction.

# Union of Conjunctive Queries

- Let  $\varphi(x_1, \dots, x_n)$  be a FOL formula with free variables  $x_1, \dots, x_n$
- $\varphi(x_1, \dots, x_n)$  is **Union of Existential Conjunctions** if it is of the form
$$\bigvee_i \psi_i(x_1, \dots, x_n)$$
where each  $\psi_i$  is an Existential Conjunction with free variables  $x_1, \dots, x_n$
- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Union of Conjunctive Queries** if  $\varphi(x_1, \dots, x_n)$  is a Union of Existential Conjunctions.

# Naïve Evaluation

Assume an (*incomplete*) database  $\mathbf{D}$  and a query  $q$

The **naïve evaluation** of  $q$  over  $\mathbf{D}$  (written  $q^\perp(\mathbf{D})$ ) is defined as follows:

1. We evaluate  $q$  over  $\mathbf{D}$  as if  $\mathbf{D}$  was a complete database
2. We discard all the tuples that contain **nulls**

**Thm:** If  $q$  is a UCQ, then  $cert(q, \mathbf{D}) = q^\perp(\mathbf{D})$

## Naïve Evaluation – Example

$$q = \{(x, y) \mid Products(x, y)\}$$

**D:** Products

| Code | Color                    |
|------|--------------------------|
| P1   | <i>Null</i> <sub>1</sub> |
| P2   | Green                    |
| P3   | Blue                     |

- Evaluate the query over the complete database:

$$q(\textcolor{brown}{D}) = \{ (P1, \textcolor{red}{Null}_1), (P2, Green), (P3, Blue) \}$$

- Discard tuples with variables:

$$q^\perp(\textcolor{brown}{D}) = \{ (P2, Green), (P3, Blue) \}$$

# Complexity of Certain Answers for UCQs

**Thm:** Let  $q$  be a UCQ. Checking whether  $\bar{a} \in cert(q, \mathbf{D})$  is NP-Complete in **Combined Complexity** and is in LOGSPACE in **Data Complexity**.

**Procedure:** Compute the **naïve evaluation**  $q^\perp(\mathbf{D})$  of  $q$  over  $\mathbf{D}$

**Corollary:** Query Answering for UCQs is in PTIME in **Data Complexity**.

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Query Answering in GAV

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. **Query answering over IISs**
9. Ontology-Based Data Management (OBDM)
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# **Query Answering in Information Integration Systems**

# Logical Formalization of Information Integration Systems – Syntax

An information integration system (IIS) is a triple  $J = \langle G, M, S \rangle$  where:

- **G** is the **global schema**
  - A logical theory over a relational alphabet  $A_G$ .
- **S** is the **source schema**
  - A logical theory over a relational alphabet  $A_S$ .
- **M** is the **mapping between S and G**
  - A logical theory over the relational alphabet  $A_S \cup A_G$ .

## Logical Formalization of I.I.S.s – Semantics

Assume a information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$

The semantics of  $\langle J, D \rangle$  is defined as follows:

$$\text{sem}(J, D) = \{ B \mid B \text{ is a } \text{global database} \text{ s.t. } (B, D) \models M \}$$

Assume a query  $q$  over  $A_G$ . The **certain answers** of  $q$  w.r.t.  $J$  and  $D$  are:

$$\text{cert}(q, J, D) = \{ \bar{a} \mid \bar{a} \in q(B) \text{ for each } B \in \text{sem}(J, D) \}$$

# Query Answering – Recognition Problem

- **DEF:** **QA-IIS** is the following decision problem:
  - **Input:** an I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , a FOL query  $q(\bar{x})$ , a source database  $\mathcal{D}$ , a tuple of constants  $\bar{c}$
  - **Question:** Is  $\bar{c} \in \text{cert}(q, J, \mathcal{D})$  ?
- **Thm:** The problem **QA-IIS** is **undecidable**.
- **Proof:** Reduction from Validity of FOL formulae.

# Conjunctive Query (recap)

- Let  $\varphi(x_1, \dots, x_n)$  be a FOL formula with free variables  $x_1, \dots, x_n$
- $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction if it is of the form

$$\exists y_1, \dots, \exists y_n P_1(\bar{z}_1) \wedge \dots \wedge P_k(\bar{z}_k)$$

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Conjunctive Query** if  $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction.

## Union of Conjunctive Queries (recap)

- A formula  $\varphi(x_1, \dots, x_n)$  is Union of Existential Conjunctions if it is of the form

$$\bigvee_i \psi_i(x_1, \dots, x_n)$$

where each  $\psi_i$  is an existential conjunction.

**Observe:** same free variables.

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Union of Conjunctive Queries** if  $\varphi(x_1, \dots, x_n)$  is a Union of Existential Conjunctions.

# Additional Assumptions

To restore decidability of query answering we assume:

- That  $\mathbf{G}$  is an empty theory over the alphabet  $A_{\mathbf{G}}$ , i.e.,  $\mathbf{G}$  is a database schema without integrity constraints.
- That  $\mathbf{S}$  is an empty theory over the alphabet  $A_{\mathbf{S}}$ , i.e.,  $\mathbf{S}$  is a database schema without integrity constraints.
- Well behaved mappings: LAV or GAV mappings defined via queries that we can actually handle: **Conjunctive Queries**

**Some small relaxations may be possible (see later)**

**From now on, if not otherwise stated, we always consider the assumptions made above to be true moreover, if not otherwise stated, we will assume that the user queries are conjunctive queries**

# (Easy) Query Answering – Recognition Problem

- A class of mapping is a family of finite logical theories (defining mappings)
- **DEF:** UCQ-QA-IIS( $\mathcal{L}$ ) where  $\mathcal{L}$  is a class of mappings is the following problem
  - **Input:** An I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  where  $\mathbf{G}$  and  $\mathbf{S}$  are empty theories and  $\mathbf{M}$  is a mapping in  $\mathcal{L}$ , a source database  $\mathbf{D}$ , a UCQ  $\mathbf{q}$ , and a tuple of constants  $\bar{c}$
  - **Question:** Is  $\bar{c} \in \text{cert}(\mathbf{q}, J, \mathbf{D})$  ?
- As we will see, UCQ-QA-IIA( $\mathcal{L}$ ) is **decidable** in several interesting cases:
  - Conjunctive GAV mapping (CQ-GAV)
  - Conjunctive LAV mapping (CQ-LAV)
  - Conjunctive GLAV mapping (CQ-LAV)

# Techniques for Query Answering

Techniques to solve **UCQ-QA-IIS( $\mathcal{L}$ )** can be divided into two families:

- **Materialization-based Approaches**
  - A special *instance* of the **global schema** is computed independently from the query
  - We compute the certain answers by evaluating the input query over such instance
  - Useful to materialize the result of the integration (**data exchange**)!
- **Virtualization-based Approaches**
  - Queries over the **global schema** are translated into suitable queries over the **source schema**
  - We compute the certain answers by evaluating the rewritten query directly over the **source database**
  - We cannot use this to materialize the result of the integration but data is always fresh!

# Query Answering In Information Integration

|     | Materialized | Virtual |
|-----|--------------|---------|
| GAV | 1            | 2       |
| LAV | 3            | 4       |

# GAV Mapping (recap)

# Global As View (recap)

GAV mappings describe the **global schema** in terms of the **source schema**

A GAV **mapping assertion** is a pair  $\langle q_s, g \rangle$  where:

- $q_s = \{\bar{x} \mid \varphi_s(\bar{x})\}$  is a FOL **query of arity n** over  $A_s$
- $g$  is a **single predicate** name of arity n in  $A_g$

Logical formalization of GAV mapping assertions

$$\forall \bar{x} . \varphi_s(\bar{x}) \rightarrow g(\bar{x})$$

A GAV mapping is a **finite** set of GAV mapping assertions.

# Conjunctive GAV (CQ-GAV)

GAV mappings describe the **global schema** in terms of the **source schema**

A **Conjunctive GAV mapping assertion** is a pair  $\langle q_S, g \rangle$  where:

- $q_S = \{\bar{x} \mid \varphi_S(\bar{x})\}$  is a **conjunctive query of arity n** over  $A_S$
- $g$  is a **single predicate** name of arity  $n$  in  $A_G$

Logical formalization of **Conjunctive GAV** mapping assertions

$$\forall \bar{x} . \varphi_S(\bar{x}) \rightarrow g(\bar{x})$$

A **Conjunctive GAV** mapping is a **finite set** of **Conjunctive GAV** mapping assertions.

# **Query Answering in GAV**

# Query Answering In Information Integration

|     | Materialized | Virtual |
|-----|--------------|---------|
| GAV | 1            | 2       |
| LAV | 3            | 4       |

# Materialization in GAV

- Assume an I.I.S  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  and a **source database  $\mathbf{D}$**  for  $\mathbf{S}$
- We want to produce a database  $\mathbf{M}(\mathbf{D})$  to represents the result of the integration
- Desired properties of the materializations.
  - We want to use the materialization  $\mathbf{M}(\mathbf{D})$  to compute **certain answers** to queries
  - We want the materialization to be as **compact** as possible
- **Idea:** to compute a materialization  $\mathbf{M}(\mathbf{D})$  of GAV mappings, we can simply materialize the views, i.e., the definitions of the atoms via the following steps:
  1. “Answer” over  $\mathbf{D}$  the **queries** in the left-hand side of each **mapping assertion**,
  2. “Load” these answers into the **elements** of the **global schemas**

## Retrieved Global Database

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a GAV information integration specification and  $D$  is a **source database** for  $S$

We call **retrieved global database** for  $J$  w.r.t.  $D$ , denoted by  $M(D)$ , the **global database** obtained by "applying" the queries in the **mapping**, and "transferring" to the elements of  $G$  the corresponding tuples retrieved from  $D$

Formally:

$$M(D) = \{g(\bar{c}) \mid \langle q_S, g \rangle \in M \text{ and } \bar{c} \in q_S(D)\}$$

Note that, since mappings are of type GAV, the tuples to be "transferred" to the **global schema  $G$**  are always definite, i.e., they do not contain existentially quantified elements.

# Retrieved Global Database: Observations

Retrieved global database for  $J$  w.r.t.  $D$ :

$$M(D) = \{g(\bar{c}) \mid \langle q_s, g \rangle \in M \text{ and } \bar{c} \in q_s(D)\}$$

In general, given a source database  $D$ , we saw that there are several **global databases** for  $G$  that satisfy  $M$  w.r.t.  $C$ . However, it is easy to prove the following properties:

**Property 1:**  $M(D) \in \text{sem}(J, D)$

**Property 2:**  $M(D) \subseteq B$  for each  $B \in \text{sem}(J, D)$

That is, the **retrieved global database** is the *smallest* (w.r.t. set containment) **global database** in  $\text{sem}(J, D)$ .

# Retrieved Global Database - Example

- **Global Schema G** containing:
  - Movie>Title, Year, Director
  - European>Director
  - Review>Title, Critique
- **Source Schema S** containing:
  - $r1(\text{Title}, \text{Year})$  : Movies since 1960 with European directors
  - $r2(\text{Title}, \text{Critique})$  : Reviews of movies since 1990
  - $r3(\text{Title}, \text{Director}, \text{Country})$  : Movies' directors with their country

## GAV Mapping $M=\{m_1, m_2, m_3\}$ :

- $m_1 = \forall t, y, d, c. r_1(t, y) \wedge r_3(t, d, c) \rightarrow \text{Movie}(t, y, d)$
- $m_2 = \forall t, d. r_3(t, d, 'EU') \rightarrow \text{European}(d)$
- $m_3 = \forall t, r. r_2(t, r) \rightarrow \text{Review}(t, r)$

# Retrieved Global Database - Example

| D     |          |         |
|-------|----------|---------|
| $r_1$ |          |         |
| Title | Year     |         |
| T1    | 1970     |         |
| T2    | 1990     |         |
| T3    | 1979     |         |
| $r_2$ |          |         |
| Title | Critique |         |
| T2    | 7/10     |         |
| $r_3$ |          |         |
| Title | Director | Country |
| T1    | D1       | EU      |
| T2    | D2       | EU      |

| Movie |      |          |
|-------|------|----------|
| Title | Year | Director |
| T1    | 1970 | D1       |
| T2    | 1990 | D2       |

$$m_1 = \forall t, y, d, c. r_1(t, y) \wedge r_3(t, d, c) \rightarrow \text{Movie}(t, y, d)$$

# Retrieved Global Database - Example

| D     |          |         |
|-------|----------|---------|
| $r_1$ |          |         |
| Title | Year     |         |
| T1    | 1970     |         |
| T2    | 1990     |         |
| T3    | 1979     |         |
| $r_2$ |          |         |
| Title | Critique |         |
| T2    | 7/10     |         |
| $r_3$ |          |         |
| Title | Director | Country |
| T1    | D1       | EU      |
| T2    | D2       | EU      |

$M(D)$

European

Director

D1

D2

$m_2 = \forall t, d. r_3(t, d, 'EU') \rightarrow European(d)$

# Retrieved Global Database - Example

| D     |          |         |
|-------|----------|---------|
| $r_1$ |          |         |
| Title | Year     |         |
| T1    | 1970     |         |
| T2    | 1990     |         |
| T3    | 1979     |         |
| $r_2$ |          |         |
| Title | Critique |         |
| T2    | 7/10     |         |
| $r_3$ |          |         |
| Title | Director | Country |
| T1    | D1       | EU      |
| T2    | D2       | EU      |

M(D)

Review

| Title | Critique |
|-------|----------|
| T2    | 7/10     |

$m_3 = \forall t, r. r_2(t, r) \rightarrow \text{Review}(t, r)$

# Retrieved Global Database - Example

| D     |          |         |
|-------|----------|---------|
| $r_1$ |          |         |
| Title | Year     |         |
| T1    | 1970     |         |
| T2    | 1990     |         |
| T3    | 1979     |         |
| $r_2$ |          |         |
| Title | Critique |         |
| T2    | 7/10     |         |
| $r_3$ |          |         |
| Title | Director | Country |
| T1    | D1       | EU      |
| T2    | D2       | EU      |

| M(D)  |      |          |
|-------|------|----------|
| Movie |      |          |
| Title | Year | Director |
| T1    | 1970 | D1       |
| T2    | 1990 | D2       |

| European |
|----------|
| Director |
| D1       |
| D2       |

| Review |          |
|--------|----------|
| Title  | Critique |
| T2     | 7/10     |

# Retrieved Global Database for Query Answering with CQ-GAV mapping

**Thm:** Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a CQ-GAV information integration specification and  $D$  is a **source database**. For any CQ  $q$  over  $G$ , we have  $\text{cert}(q, J, D) = q(M(D))$

Exercise: Prove the Theorem

*Hints:* First prove that  $\text{cert}(q, J, D) \subseteq q(M(D))$  by contraposition, i.e. prove that  $\bar{c} \notin q(M(D))$  implies  $\bar{c} \notin \text{cert}(q, J, D)$  using **Property 1**

Then, prove that  $q(M(D)) \subseteq \text{cert}(q, J, D)$  using **Property 2** + the fact that CQs are *monotone*, i.e.  $q(B) \subseteq q(B')$  holds whenever  $q$  is a CQ and  $B \subseteq B'$

# Retrieved Global Database for Query Answering with CQ-GAV mapping

**Thm:** Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a CQ-GAV information integration specification and  $D$  is a **source database**. For any CQ  $q$  over  $G$ , we have  $\text{cert}(q, J, D) = q(M(D))$

**Proof:** Suppose  $\bar{c} \notin q(M(D))$ . Since  $M(D) \in \text{sem}(J, D)$  (cf. **Property 1**), we have  $\bar{c} \notin \text{cert}(q, J, D)$  by definition of certain answers.

Suppose now  $\bar{c} \in q(M(D))$ . Since  $M(D) \subseteq B$  for each  $B \in \text{sem}(J, D)$  (cf. **Property 2**), and since CQs are *monotone*, i.e.  $q(B) \subseteq q(B')$  holds for any  $B \subseteq B'$ , we derive  $\bar{c} \in q(B)$  for any  $B \in \text{sem}(J, D)$ . Hence, we have that  $\bar{c} \in \text{cert}(q, J, D)$ .

Can we extend these results?

# Retrieved Global Database for Query Answering with FOL-GAV mapping

Actually the theorem holds in case of general GAV mapping (FOL-GAV)!

**Thm:** Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a **FOL-GAV** information integration specification and  $D$  is a **source database**. For any CQ  $q$  over  $G$ , we have  $\text{cert}(q, J, D) = q(M(D))$ .

**Proof:** the same technique can be used also in case of general GAV mapping (FOL-GAV). The claim then follows from the fact that the retrieved global database has the same exact properties as in the case of CQ-GAV.

# Query Answering In Information Integration

|     | Materialized | Virtual |
|-----|--------------|---------|
| GAV | 1            | 2       |
| LAV | 3            | 4       |

# Virtualization in GAV

- We want to produce a query over the **sources** that captures the results of a user query over the **global schema**.
- Desired properties of the source query.
  - We want to use it to compute **certain answers**.
  - It should fall into a specific query language.
  - It should have the right size.
- **Virtualization in GAV is very simple:** we replace atoms in the user query with their definition in the **mapping**.

# Perfect Rewritings

- Assume an Information Integration System  $J = \langle \textcolor{blue}{G}, \textcolor{green}{M}, \textcolor{brown}{S} \rangle$ .
  - No further assumptions on the three components for now
- Assume a class of queries  $L$  (a language of queries), e.g., CQs, UCQs, FOL, ...
- **Definition:** Given a query  $\textcolor{blue}{q}$  over  $\textcolor{blue}{G}$ , a **perfect  $L$ -rewriting of  $\textcolor{blue}{q}$  w.r.t.  $J$**  is a query  $\textcolor{blue}{rew}_L^{(\textcolor{blue}{q}, J)}$  in the language  $L$  such that the following condition holds for every database  $\textcolor{brown}{D}$  for  $\textcolor{brown}{S}$

$$\bar{a} \in \text{cert}(\textcolor{blue}{q}, J, \textcolor{brown}{D}) \text{ if and only if } \bar{a} \in \textcolor{blue}{rew}_L^{(\textcolor{blue}{q}, J)}(\textcolor{brown}{D})$$

# Query Unfolding in CQ-GAV

The **unfolding of a query  $q$  over  $\mathbf{G}$  w.r.t.  $\mathbf{M}$**  (denoted by  $unf_{q,M}$ ) is the query over  $\mathbf{S}$  obtained from  $q$  by substituting every atom  $R(\bar{z})$  in  $q$  such that  $\langle \psi_S, R \rangle \in M$  with the corresponding definition  $\psi_S(\bar{z})$ .

The **unfolding of a UCQ** is the union of the unfolding of all its disjuncts.

- **Thm:** Given an information integration specification  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , where  $\mathbf{M}$  is a CQ-GAV mapping, and given a UCQ  $q$  over  $\mathbf{G}$ , we have that  $unf_{q,M}$  is a **perfect UCQ-rewriting** of  $q$  w.r.t.  $J$ .
- **Corollary:** For any UCQ  $q$  over  $\mathbf{G}$ , and **source database  $D$  for  $S$** , we have  $cert(q,J,D) = unf_{q,M}(D)$

# Query Unfolding – Example (1\3)

- **Global Schema G** containing:
  - Paper(ID, Title)
  - Author(PaperID, authorID, position)
  - Researcher(ID, Affiliation)
- **Source Schema S** containing:
  - **T1(article, title, SSD)** : papers with their title and scientific sectors (e.g., chemistry, mathematic, etc.).
  - **T2(article, author, university, pos)** : authors of scientific articles with their affiliation and position as author of the article.
  - **T3(researcher, univ, year)** : affiliation and affiliation date of the researchers.
- **GAV Mapping M={m<sub>1</sub>,m<sub>2</sub>,m<sub>3</sub>, m<sub>4</sub>}**:
  - **m<sub>1</sub>=**  $\forall a, t, s. T1(a, t, s) \rightarrow \text{Paper}(a, t)$
  - **m<sub>2</sub>=**  $\forall ar, at, u, p. T2(ar, at, u, p) \rightarrow \text{Author}(ar, at, p)$
  - **m<sub>3</sub>=**  $\forall ar, at, u, p. T2(ar, at, u, p) \rightarrow \text{Researcher}(at, u)$
  - **m<sub>4</sub>=**  $\forall r, u, y. T3(r, u, y) \rightarrow \text{Researcher}(t, u)$

## Query Unfolding – Example (2\3)

- **GAV Mapping M={m<sub>1</sub>,m<sub>2</sub>,m<sub>3</sub>, m<sub>4</sub>}**:

- m<sub>1</sub>=  $\forall a, t, s. T1(a,t,s) \rightarrow \text{Paper}(a,t)$
- m<sub>2</sub>=  $\forall ar, at, u, p. T2(ar,at,u,p) \rightarrow \text{Author}(ar,at,p)$
- m<sub>3</sub>=  $\forall ar, at, u, p. T2(ar,at,u,p) \rightarrow \text{Researcher}(at,u)$
- m<sub>4</sub>=  $\forall r, u, y. T3(r,u,y) \rightarrow \text{Researcher}(t,u)$

User query over **G** asking for "*the papers together with their authors and affiliation*":

$$q = \{(p, r, a) \mid \exists n. \text{Researcher}(r,a) \wedge \text{Author}(p,r,n)\}$$

*unf*<sub>q,M</sub> is the union of the following two CQs over **S**:

- $\{(\textcolor{blue}{p}, \textcolor{violet}{r}, \textcolor{red}{a}) \mid \exists x, y, z, k. T2(x, \textcolor{violet}{r}, \textcolor{red}{a}, y) \wedge T2(\textcolor{blue}{p}, \textcolor{violet}{r}, z, k)\}$
- $\{(\textcolor{blue}{p}, \textcolor{violet}{r}, \textcolor{red}{a}) \mid \exists x, y, z. T3(\textcolor{violet}{r}, \textcolor{red}{a}, x) \wedge T2(\textcolor{blue}{p}, \textcolor{violet}{r}, y, z)\}$

## Query Unfolding – Example (3\3)

- **GAV Mapping M={m<sub>1</sub>,m<sub>2</sub>,m<sub>3</sub>, m<sub>4</sub>}**:

- m<sub>1</sub>=  $\forall a, t, s. T1(a,t,s) \rightarrow \text{Paper}(a,t)$
- m<sub>2</sub>=  $\forall ar, at, u, p. T2(ar,at,u,p) \rightarrow \text{Author}(ar,at,p)$
- m<sub>3</sub>=  $\forall ar, at, u, p. T2(ar,at,u,p) \rightarrow \text{Researcher}(at,u)$
- m<sub>4</sub>=  $\forall r, u, y. T3(r,u,y) \rightarrow \text{Researcher}(t,u)$

User query over **G** asking for "*the papers together with their authors and affiliation*":

$$q = \{(p, r, a) \mid \exists n. \text{Researcher}(r,a) \wedge \text{Author}(p,r,n)\}$$

*unf*<sub>q,M</sub> is the union of the following two CQs over **S**:

- $\{(\textcolor{blue}{p}, \textcolor{purple}{r}, \textcolor{red}{a}) \mid \exists x, y, z, k. \textcolor{brown}{T2(x, r, a, y)} \wedge \textcolor{brown}{T2(p, r, z, k)}\}$
- $\{(\textcolor{blue}{p}, \textcolor{purple}{r}, \textcolor{red}{a}) \mid \exists x, y, z. \textcolor{brown}{T3(r, a, x)} \wedge \textcolor{brown}{T2(p, r, y, z)}\}$

Unfolding

# FOL-GAV Mappings

Unfolding can be used also with more expressive (FOL-)GAV mappings!

- In the case of general GAV mappings the algorithm works as before
- **Thm:** Given an information integration specification  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , where  $\mathbf{M}$  is a FOL-GAV mapping, and given a UCQ  $\mathbf{q}$  over  $\mathbf{G}$ , we have that  $unf_{\mathbf{q}, \mathbf{M}}$  is a **perfect FOL-rewriting** of  $\mathbf{q}$  w.r.t.  $J$ .

**Observe:** in this case,  $unf_{\mathbf{q}, \mathbf{M}}$  is a FOL query instead of a UCQ

# Query Answering – Recognition Problem

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a CQ-GAV information integration specification (i.e.,  $M$  is a CQ-GAV mapping) and  $D$  is a **source database**.

We know that  $unf_{q,M}$  is a UCQ that is the **perfect UCQ-rewriting** of  $q$  w.r.t.  $J$  and so we have that a tuple of constants  $\bar{a} \in cert(q, J, D)$  if and only if  $\bar{a} \in unf_{q,M}(D)$ . From these observations, and from the fact that query answering of a UCQ (FOL-query) over a database is in LogSpace, we have:

**Thm:** UCQ-QA-IIS(CQ-GAV) is polynomial in **data complexity**, actually LogSpace.

This result holds also in case of the FOL-GAV class of mapping, i.e., UCQ-QA-IIS(FOL-GAV) is polynomial (in LogSpace) in **data complexity**.

# Materialization vs Virtualization in GAV

- The size of the Retrieved Global Database depends exponentially on the size of the arity of predicates in the global schema!
- It is very easy to compute the unfolding.
- Should we always use virtualization?
  - Do we want to save a snapshot of the data?
  - Is accessing sources as easy as accessing a retrieved global database?
- The answer to the above question may depend on the scenario of use.

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Query Answering in (G)LAV Materialization

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. **Query answering over IISs**
9. Ontology-Based Data Management (OBDM)
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# **Query Answering in (G)LAV**

# Logical Formalization of Information Integration Systems – Syntax

An information integration system (IIS) is a triple  $J = \langle G, M, S \rangle$  where:

- **G** is the **global schema**
  - A logical theory over a relational alphabet  $A_G$ .
- **S** is the **source schema**
  - A logical theory over a relational alphabet  $A_S$ .
- **M** is the **mapping between S and G**
  - A logical theory over the relational alphabet  $A_S \cup A_G$ .

## Logical Formalization of I.I.S.s – Semantics

Assume a information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$

The semantics of  $\langle J, D \rangle$  is defined as follows:

$$\text{sem}(J, D) = \{ B \mid B \text{ is a } \text{global database} \text{ s.t. } (B, D) \models M \}$$

Assume a query  $q$  over  $A_G$ . The **certain answers** of  $q$  w.r.t.  $J$  and  $D$  are:

$$\text{cert}(q, J, D) = \{ \bar{a} \mid \bar{a} \in q(B) \text{ for each } B \in \text{sem}(J, D) \}$$

# Query Answering – Recognition Problem

- **DEF:** **QA-IIS** is the following decision problem:
  - **Input:** an I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , a FOL query  $q(\bar{x})$ , a source database  $\mathcal{D}$ , a tuple of constants  $\bar{c}$
  - **Question:** Is  $\bar{c} \in \text{cert}(q, J, \mathcal{D})$  ?

# Conjunctive Query (recap)

- Let  $\varphi(x_1, \dots, x_n)$  be a FOL formula with free variables  $x_1, \dots, x_n$
- $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction if it is of the form

$$\exists y_1, \dots, \exists y_n P_1(\bar{z}_1) \wedge \dots \wedge P_k(\bar{z}_k)$$

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Conjunctive Query** if  $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction.

## Union of Conjunctive Queries (recap)

- A formula  $\varphi(x_1, \dots, x_n)$  is Union of Existential Conjunctions if it is of the form

$$\bigvee_i \psi_i(x_1, \dots, x_n)$$

where each  $\psi_i$  is an existential conjunction.

**Observe:** same free variables.

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Union of Conjunctive Queries** if  $\varphi(x_1, \dots, x_n)$  is a Union of Existential Conjunctions.

# Our Assumptions

As made for GAV, we assume:

- That **G** is an empty theory over the alphabet  $A_G$ , i.e., **G** is a database schema without integrity constraints.
- That **S** is an empty theory over the alphabet  $A_S$ , i.e., **S** is a database schema without integrity constraints.
- That **GLAV mappings** are defined via queries that we can actually handle: **Unions of Conjunctive Queries**

**From now on, if not otherwise stated, we always consider the assumptions made above to be true moreover, if not otherwise stated, we will assume that the user queries are conjunctive queries**

## Local As View (recap)

LAV mappings describe the **source schema** in terms of the **global schema**

A LAV **mapping assertion** is a pair  $\langle s, q_g \rangle$  where:

- $s$  is a single predicate name of arity  $n$  in  $A_S$
- $q_g = \{\bar{x} \mid \varphi_g(\bar{x})\}$  is a FOL **query of arity n over  $A_G$**

Logical formalization of LAV mapping assertions:

$$\forall \bar{x} . s(\bar{x}) \rightarrow \varphi_g(\bar{x})$$

A LAV mapping is a **finite** set of LAV mapping assertions

# Global and Local As View (recap)

Materialization techniques for **LAV** mapping are similar to those for **GLAV**, so we will look at the more general case of **GLAV**

A **GLAV mapping assertion** is a pair  $\langle \mathbf{q}_s, \mathbf{q}_g \rangle$  where:

- $\mathbf{q}_s = \{\bar{x} \mid \varphi_s(\bar{x})\}$  is a FOL query of arity **n** over  $\mathcal{A}_S$
- $\mathbf{q}_g = \{\bar{x} \mid \varphi_g(\bar{x})\}$  is a FOL query of arity **n** over  $\mathcal{A}_G$

Logical formalization of GLAV mapping assertions:

$$\forall \bar{x}. \mathbf{\varphi}_s(\bar{x}) \rightarrow \mathbf{\varphi}_g(\bar{x})$$

A GLAV mapping is a **finite** set of GLAV mapping assertions.

# Conjunctive GLAV Mapping

A **Conjunctive GLAV mapping assertion** is a pair  $\langle \mathbf{q}_s, \mathbf{q}_g \rangle$  where:

- $\mathbf{q}_s = \{\bar{x} \mid \varphi_s(\bar{x})\}$  is a **CQ** of arity **n** over  $\mathcal{A}_S$
- $\mathbf{q}_g = \{\bar{x} \mid \varphi_g(\bar{x})\}$  is a **CQ** of arity **n** over  $\mathcal{A}_G$

A **Conjunctive GLAV mapping** is a **finite** set of Conjunctive GLAV mapping assertions.

# Conjunctive Mapping – Examples (1)

- Conjunctive **LAV** mapping

$$\forall x, y. \textcolor{brown}{Tab1}(x, y) \rightarrow \exists z. \textcolor{blue}{Person}(x) \wedge \textcolor{blue}{Parent}(y) \wedge \textcolor{blue}{hasFather}(x, z)$$

- Conjunctive **GAV** mapping

$$\forall x. \forall y. \exists z. \textcolor{brown}{Tab1}(x, y) \wedge \textcolor{brown}{Tab2}(y, z) \rightarrow \textcolor{blue}{hasFather}(x, y)$$

- Conjunctive **GLAV** mapping

$$\forall x. \forall y. \exists z. \textcolor{brown}{Tab1}(x, y) \wedge \textcolor{brown}{Tab2}(y, z) \rightarrow \exists v. \textcolor{blue}{hasFather}(x, y) \wedge \textcolor{blue}{hasFather}(y, v)$$

## Conjunctive Mapping – Examples (2)

- Non-Conjunctive LAV mapping

$$\forall x, y. \textcolor{brown}{Tab1}(x, y) \rightarrow \exists z. \textcolor{blue}{Person}(x) \wedge \textcolor{blue}{Parent}(y) \wedge \textcolor{blue}{hasFather}(x, z) \wedge \neg \textcolor{blue}{Woman}(z)$$


- Non-Conjunctive GAV mapping

$$\forall x. \forall y. \exists z. \textcolor{brown}{Tab1}(x, y) \wedge \textcolor{brown}{Tab2}(y, z) \wedge \neg \textcolor{brown}{Tab3}(z) \rightarrow \textcolor{blue}{hasFather}(x, y)$$


- Non-Conjunctive GLAV mapping

$$\forall x. \forall y. \textcolor{brown}{Tab1}(x, y) \rightarrow \textcolor{blue}{hasFather}(x, y) \vee \textcolor{blue}{hasMother}(x, y)$$


# Query Answering – Recognition Problem

- Let **CQ-GLAV** be the class of **conjunctive GLAV mappings**
- **DEF:** **UCQ-QA-IIS( $\mathcal{L}$ )** where  $\mathcal{L}$  is a class of mappings is the following problem
  - **Input:** An I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  where  $\mathbf{G}$  and  $\mathbf{S}$  are empty theories and  $\mathbf{M}$  is a mapping in  $\mathcal{L}$ , a source database  $\mathbf{D}$ , a UCQ  $\mathbf{q}$ , and a tuple of constants  $\bar{c}$
  - **Question:** Is  $\bar{c} \in \text{cert}(\mathbf{q}, J, \mathbf{D})$  ?
- **Thm:** **UCQ-QA(CQ-GLAV)** is **decidable**
  - Via materialization
  - Via virtualization

# Query Answering In Information Integration

|     | Materialized | Virtual |
|-----|--------------|---------|
| GAV | 1            | 2       |
| LAV | 3            | 4       |

# Materialization in (G)LAV

Assume an I.I.S  $J = \langle G, M, S \rangle$  and a **source database D** for **S**

As for GAV, we want to produce a database **M(D)** to represents the result of the integration, i.e., a **global instance** representing all the **global databases** in  $\text{sem}(J, D)$

Desired properties of the materializations.

- We want to use the materialization **M(D)** to compute **certain answers** to queries
- We want the materialization to be as **compact** as possible

With (G)LAV we cannot simply materialize views from the **sources** since there is no direct correspondence between definitions in the mapping and **global predicates**. Moreover, **we need to deal with existential quantifiers on the right-hand side of mapping assertions!**

## Materialization in (G)LAV

With (G)LAV we cannot simply materialize views from the sources

- There is no direct correspondence between definitions and global predicates.
- Missing information (existential quantification) in the global schema definition

Intuitively, to materialize GLAV mappings we need **null** values

A **global instance  $K$**  over  $G$  is like a **global database** over  $G$  except that it may contain also **labeled nulls** (we can think of such **labeled nulls** simply as placeholders for *unknown* constants).

We denote by  $\text{Const}(K)$  (resp.  $\text{Nulls}(K)$ ) the set of constants (resp. labeled nulls) occurring in  $K$

With a slight abuse of notation, we let  $\text{dom}(K) = \text{Const}(K) \cup \text{Nulls}(K)$

# Materialization in LAV – Example

Source database:  $D = \{R(a,b), R(b,c)\}$

Mapping:  $M = \{\forall x, y. R(x,y) \rightarrow \exists z. S(x,z) \wedge S(z,y)\}$

Which of the following **global instances** would be a good materialization?

- Symbols  $x$ ,  $y$ , and  $z$  denote **labeled nulls**
- $K_1 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,d), S(d,c)\}$
- $K_2 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b)\}$
- $K_3 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(a,\textcolor{red}{y}), S(\textcolor{red}{y},b), S(b,\textcolor{red}{z}), S(\textcolor{red}{z},c)\}$
- $K_4 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,\textcolor{red}{y}), S(\textcolor{red}{y},c)\}$
- $K_5 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,\textcolor{red}{x}), S(\textcolor{red}{x},c)\}$

We still do not know!

# Homomorphisms

# A new tool: the homomorphism

To prove the next results we will rely on the notion of **homeomorphism**

Homomorphisms are a class of functions between relational structures

- Relational Structures are, essentially, sets of relational atoms
- A special instance of what mathematicians call structure-preserving mappings, i.e., mappings that do not alter the shape of a structure “too much”

We borrow the notion of homomorphisms and tailor it to our needs.

- **Don't tell mathematicians!**

# The notion of homomorphism

Let **A** and **B** be two database instances over the same schema

A **homomorphism** from **A** to **B** is a function  $h$  from  $\text{dom}(\mathbf{A})$  to  $\text{dom}(\mathbf{B})$   
( $\text{dom}(A)$  and  $\text{dom}(B)$  can contain both *constants* and *variables\labeled nulls*)  
satisfying the following properties:

1. For every constant  $c \in \text{Const}(\mathbf{A})$ ,  $h(c) = c$
2. For every  $R(\bar{a}) \in \mathbf{A}$ , we have  $R(h(\bar{a})) \in \mathbf{B}$

For a tuple  $\bar{a} = (a_1, \dots, a_n)$  of constants and labeled nulls,  $h(\bar{a}) = (h(a_1), \dots, h(a_n))$ . In other words, condition 2. always ensures  $h(\mathbf{A}) \subseteq \mathbf{B}$  whenever  $h$  is a **homomorphism** from **A** to **B**, where  $h(\mathbf{A}) = \{R(h(\bar{a})) \mid R(\bar{a}) \in \mathbf{A}\}$

## Homomorphisms – Example

**Example:**  $\text{Const}(A) = \{a\}$  and  $\text{Nulls}(A) = \{x,y\}$

$\text{Const}(B) = \{a,b,c\}$  and  $\text{Nulls}(B) = \{\}$

$$\begin{array}{ccc} A = \{ R(\textcolor{brown}{a}, \textcolor{blue}{x}), & & S(\textcolor{blue}{x}, \textcolor{blue}{y}) \} \\ \downarrow & & \downarrow \\ B = \{ R(\textcolor{brown}{a}, \textcolor{brown}{b}), \ R(\textcolor{brown}{b}, \textcolor{brown}{b}), \ S(\textcolor{brown}{b}, \textcolor{brown}{c}) \} & & \end{array}$$

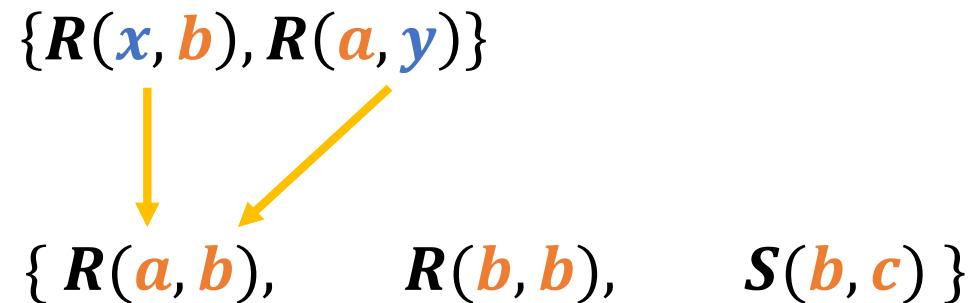
The following function  $h$  defines a homomorphism from  $A$  to  $B$

- $h(a) = a$
- $h(x) = b$
- $h(y) = c$

## Homomorphisms – Example

**Example:**  $\text{Const}(A) = \{a,b\}$  and  $\text{Nulls}(A) = \{x,y\}$

$\text{Const}(B) = \{a,b,c\}$  and  $\text{Nulls}(B) = \{\}$



The following function  $h$  defines a homomorphism

- $h(a) = a, h(b) = b$
- $h(x) = a$
- $h(y) = b$

## Homomorphisms – Example

**Example:**  $\text{Const}(A) = \{a,b\}$  and  $\text{Nulls}(A) = \{x,y\}$

$\text{Const}(B) = \{a,b,c\}$  and  $\text{Nulls}(B) = \{\}$

$$\{ R(\textcolor{brown}{a}, \textcolor{blue}{x}), \quad S(\textcolor{blue}{x}, \textcolor{brown}{b}), \quad S(\textcolor{blue}{x}, y) \}$$

$$\{ R(\textcolor{brown}{a}, \textcolor{brown}{b}), \quad R(\textcolor{brown}{b}, \textcolor{brown}{b}), \quad S(\textcolor{brown}{b}, c) \}$$

**There is no homomorphisms here!**

## Exercise on Homomorphisms

For which of the following there exists a homomorphism? In red the labeled nulls

1.  $A = \{R(a,b)\}$

$B = \{R(a,b)\}$

2.  $A = \{R(a,\textcolor{red}{x})\}$

$B = \{R(c,d)\}$

3.  $A = \{R(a,\textcolor{red}{x}), S(\textcolor{red}{x},c)\}$

$B = \{R(a,b), R(d,c)\}$

4.  $A = \{R(a,\textcolor{red}{x})\}$

$B = \{R(a,b), R(b,c)\}$

# Exercise on Homomorphisms - Solution

For which of the following there exists a homomorphism? In red the labeled nulls

1.  $A = \{R(a,b)\}$

– Yes. Identity.

$B = \{R(a,b)\}$

2.  $A = \{R(a,\textcolor{red}{x})\}$

– No.

$B = \{R(c,d)\}$

3.  $A = \{R(a,\textcolor{red}{x}), S(\textcolor{red}{x},c)\}$

– No.

$B = \{R(a,b), R(d,c)\}$

4.  $A = \{R(a,\textcolor{red}{x})\}$

– Yes,  $h(x) = b$ .

$B = \{R(a,b), R(b,c)\}$

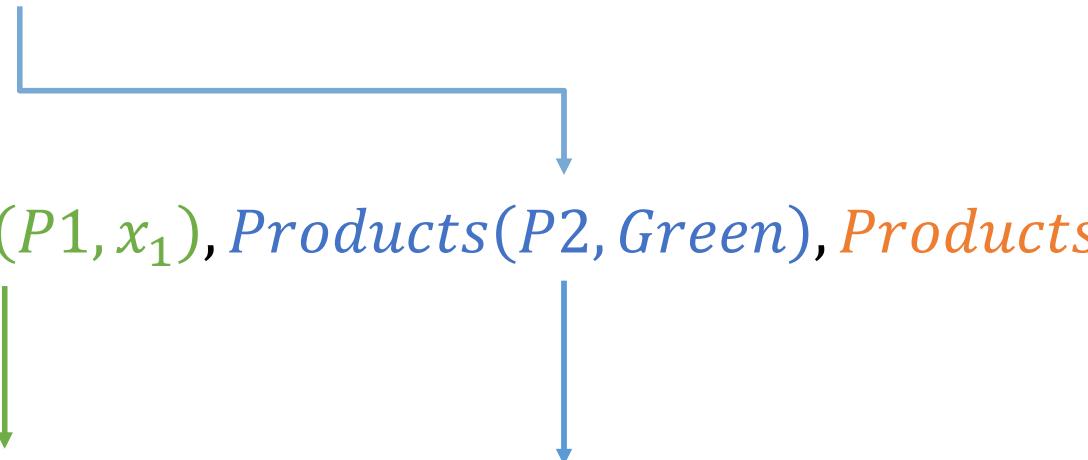
# Composition of Homomorphisms

Let  $h$  be an homomorphism from  $A$  to  $B$ , and let  $h'$  be an homomorphism from  $B$  to  $C$ . The **composition**  $h' \circ h$  of  $h, h'$  is the mapping defined as follows:  $h' \circ h(x) = h'(h(x))$

**Thm:** The composition  $h' \circ h$  is itself a homomorphism  $h''$  from  $A$  to  $C$ .

**Proof:** Let  $\alpha \in A$ . Then,  $h(\alpha) \in B$  and therefore  $h'(h(\alpha)) \in C$ . The claim follows straightforwardly.

# Composition of Homomorphisms – Example

- $\{Products(x, Green)\}$ 
    - $\{Products(P1, x_1), Products(P2, Green), Products(P3, Blue)\}$ 
      - $\{Products(P1, Red), Products(P2, Green), Products(P3, Blue)\}$
- 

# Finding Homomorphisms

Finding **homomorphisms** is harder than it may seem

- **Thm:** Given two sets of atoms A and B, checking whether there exists a homomorphism from A to B is NP-Complete
- **Thm:** Let A be a set of atoms. For an input set of atoms B, checking whether there exists a homomorphism from A to B is in LOGSPACE
  - Indeed, if we fix A, the number of possible assignments we need to test becomes small

# Homomorphisms – Extensions

We can extend the notion of **homomorphisms** to different objects

- **Complete Databases.** Given an interpretation  $I$ , i.e., a complete database, we can naturally convert  $I$  into a set of atoms.
- **Incomplete Databases.** An incomplete database is a logical theory (set) of relational atoms.
- **Conjunctive Queries.** A conjunction of atoms can be seen as a set.

From now on, we will talk about **homomorphisms** in all the cases above

# Query Answering and Homomorphisms (Complete Databases)

We have following well known property:

Let  $q = \{\bar{x} \mid \bigvee_i \psi_i(x_1, \dots, x_n)\}$  be a UCQ and  $D$  be a database instance

**Thm:** a tuple  $\bar{a} \in q(D)$  if and only if there exists  $i = 1, \dots, n$  and an **homomorphism**  $h$  from  $\psi_i$  to  $D$  such that  $h(\bar{x}) = \bar{a}$

**Let's go back to our question:**

**What **global instance** is a good materialization?**

# Solutions and Universal Solutions

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a GLAV information integration specification and  $D$  is a **source database**

A **global instance**  $K$  over  $G$  is like a **global database** over  $G$  except that it may contain also **labeled nulls**

- **Definition:** A **global instance**  $K$  is a **solution** for  $J$  w.r.t.  $D$  if  $q_S(D) \subseteq q_G(K)$  for each mapping assertion  $m = \langle q_S, q_G \rangle$  in  $M$ , i.e.,  $(K, D) \models M$
- **Definition:** A **global instance**  $K$  is a **universal solution** for  $J$  w.r.t.  $D$  if  $K$  is a solution for  $J$  w.r.t.  $D$  such that, for each  $B \in \text{sem}(J, D)$ , there exists a **homomorphism** from  $K$  to  $B$

# Exercise

Source database:  $D = \{R(a,b), R(b,c)\}$

Mapping:  $M = \{\forall x, y. R(x,y) \rightarrow \exists z. S(x,z) \wedge S(z,y)\}$

Which of the following **global instances** is a **solution** for  $J = \langle G, M, S \rangle$  w.r.t.  $D$ ? Which is a **universal solution** for  $J$  w.r.t.  $D$ ?

➤ Symbols  $x$ ,  $y$ , and  $z$  denote **labeled nulls**

- $K_1 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,d), S(d,c)\}$
- $K_2 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b)\}$
- $K_3 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(a,\textcolor{red}{y}), S(\textcolor{red}{y},b), S(b,\textcolor{red}{z}), S(\textcolor{red}{z},c)\}$
- $K_4 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,\textcolor{red}{y}), S(\textcolor{red}{y},c)\}$
- $K_5 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,\textcolor{red}{x}), S(\textcolor{red}{x},c)\}$

# Exercise

Source database:  $D = \{R(a,b), R(b,c)\}$

Mapping:  $M = \{\forall x, y. R(x,y) \rightarrow \exists z. S(x,z) \wedge S(z,y)\}$

Which of the following **global instances** is a **solution** for  $J = \langle G, M, S \rangle$  w.r.t.  $D$ ? Which is a **universal solution** for  $J$  w.r.t.  $D$ ?

- Symbols  $x$ ,  $y$ , and  $z$  denote **labeled nulls**
- $K_1 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,d), S(d,c)\}$  - **Solution but not universal**
- $K_2 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b)\}$  - **Not a Solution**
- $K_3 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(a,\textcolor{red}{y}), S(\textcolor{red}{y},b), S(b,\textcolor{red}{z}), S(\textcolor{red}{z},c)\}$  - **Universal Solution**
- $K_4 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,\textcolor{red}{y}), S(\textcolor{red}{y},c)\}$  - **Universal Solution**
- $K_5 = \{S(a,\textcolor{red}{x}), S(\textcolor{red}{x},b), S(b,\textcolor{red}{x}), S(\textcolor{red}{x},c)\}$  - **Solution but not universal**

## Naïve Evaluation (recap)

Assume an (*incomplete*) database  $\mathbf{D}$  and a query  $q$ . We denote by  $q(\mathbf{D})$  the evaluation of  $q$  over  $\mathbf{D}$ .

The **naïve evaluation** of  $q$  over  $\mathbf{D}$  (written  $q^\perp(\mathbf{D})$ ) is defined as follows:

1. We evaluate  $q$  over  $\mathbf{D}$  as if  $\mathbf{D}$  was a complete database
2. We discard all the tuples that contain **labelled nulls**

That is,  $q^\perp(\mathbf{D})$  is a function that removes from  $q(\mathbf{D})$  all the tuples containing at least a **labeled null**

# Certain Answers via Universal Solutions

Consider an information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a GLAV information integration specification and  $D$  is a **source database**

**Theorem U:** Let  $K$  be a universal solution for  $J$  w.r.t.  $D$ . For any CQ  $q$ , we have  $\text{cert}(q, J, D) = q^\perp(K)$

**Proof (sketch):** We first prove that  $\text{cert}(q, J, D) \subseteq q^\perp(K)$  by contraposition, i.e. by proving that  $\bar{c} \notin q^\perp(K)$  implies  $\bar{c} \notin \text{cert}(q, J, D)$  (by the notion of universal solution). Then, we prove that  $q^\perp(K) \subseteq \text{cert}(q, J, D)$  by using the following given property: for any CQ  $q = \{\bar{x} \mid \phi(\bar{x})\}$  and instance  $K$ ,  $\bar{c} \in q(K)$  iff there is a homomorphism  $p$  from the set of atoms in  $\phi(\bar{x})$  to  $K$  such that  $p(\bar{x}) = \bar{c}$ . So, suppose  $\bar{c} \in q^\perp(K)$ . We know that there is a homomorphism  $h$  from the set of atoms in  $\phi(\bar{x})$  to  $K$  such that  $h(\bar{x}) = \bar{c}$ . Consider any  $B \in \text{sem}(J, D)$ . Since there is a homomorphism  $p$  from  $K$  to  $B$ , the composition function  $p \circ h$  is a homomorphism from the set of atoms in  $\phi(\bar{x})$  to  $B$  such that  $h(\bar{x}) = \bar{c}$ , and therefore  $\bar{c} \in q(B)$ . This implies that  $\bar{c} \in \text{cert}(q, J, D)$ .

## Universal Solutions – Some Observations

The definition of Universal Solutions does not guarantee existence. Indeed, the definition is not **constructive** and does not provide a procedure to compute it.

Universal solutions may not be unique. They are homomorphically equivalent, i.e., for every pair of universal solutions  $\mathbf{K}$  and  $\mathbf{K}'$  there exist homomorphisms  $h$  from  $\mathbf{K}$  to  $\mathbf{K}'$  and  $h'$  from  $\mathbf{K}'$  to  $\mathbf{K}$ .

For **GAV** mappings, the retrieved **global database** is a **universal solution!**

- It satisfies all the mappings
- It is contained in every model, therefore the identity is the desired homomorphism!

For **GLAV** mappings, we can use the **chase** algorithm(s): a family of algorithms that compute **universal solutions**.

# The Chase

The **chase** is a family of algorithms that can compute universal solutions.

- In the literature there are different versions of this algorithm, we will see the simplest.

The chase can be used to perform reasoning tasks over existential rules

**Definition:** An **existential rule** is a FOL formula of the following form where  $\phi$  and  $\psi$  are conjunctions of atoms.

$$\forall \bar{x}. \forall \bar{y}. \phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}. \psi(\bar{x}, \bar{z})$$

The chase is based on a set of **intuitions** that we discuss in the next slides.

# The Chase – Triggerable Mappings

Assume a GLAV mapping  $\mathbf{M}$  and a database  $\mathbf{D}$ .

**Def:** A mapping assertion  $\langle q_S, q_G \rangle \in \mathbf{M}$  is **triggerable over  $\mathbf{D}$**  for a tuple of constants  $\bar{a}$  (called the trigger) if  $\bar{a} \in q_S(\mathbf{D})$

**Example:** consider a database  $\mathbf{D} = \{R(a, a), R(b, c)\}$  and the mapping  $\mathbf{M} = \{m_1 : \forall x. R(x, x) \rightarrow \exists y. S(x, y)\}$

We have that  $m_1$  is **triggerable** for  $\langle a, a \rangle$  but not for  $\langle b, c \rangle$

## Triggerable Mappings – Intuition

**Intuition 1:** to build a solution we need to “materialize” all **triggerable mappings** (for each possible tuple that is a **trigger**)

**Ex:**  $D = \{R(a, a), R(b, c)\}$

$M = \{m_1 : \forall x. R(x, x) \rightarrow \exists y. S(x, y)\}$

We have that  $m_1$  is **triggerable** for  $\langle a, a \rangle$

**Ex:**  $K = \{S(a, b), S(a, a), S(c, d)\}$  is a solution.

# The Chase – Materializing a GLAV Mapping

**Intuition 2:** In order to “materialize” a triggerable mapping assertion we can use **variables** (a.k.a. labeled nulls) to fill up the ***existentially quantified attributes***

- We do not need to **invent constants** for the existential variables!

**Ex:**  $D = \{R(a, a), R(b, c)\}$

$M = \{m_1 : \forall x. R(x, x) \rightarrow \exists y. S(x, y)\}$

We have that  $m_1$  is **triggerable** for  $\langle a, a \rangle$

**Ex:**  $K = \{S(a, x), S(a, y), S(c, d)\}$  is a solution.

# The Chase – Smallest Materialization

**Intuition 3:** To compute the most general (**universal**) solution we need to materialize all and **only** the triggerable mappings in the **most general way!**

- To define a universal solution, we cannot add additional information.

**Ex:**  $D = \{R(a, a), R(b, c)\}$

$M = \{m_1 : \forall x. R(x, x) \rightarrow \exists y. S(x, y)\}$

We have that  $m_1$  is **triggerable** for  $\langle a, a \rangle$

**Ex:**  $K = \{S(a, x)\}$  is an universal solution.

# The Naïve Chase Algorithm

**Input:**  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a GLAV information integration specification and  $D$  is a **source database**

**Output:** A **global instance K** over  $G$  (i.e., an incomplete database over  $G$ )

$K \leftarrow \emptyset$

**For each**  $m : \forall \bar{x}. \phi(\bar{x}) \rightarrow \exists \bar{y}. \psi(\bar{x}, \bar{y})$  occurring in  $M$  **do**:

**For each**  $\bar{c} \in q_s(D)$ , where  $q_s = \{\bar{x} \mid \phi(\bar{x})\}$  **do**:

**For any**  $y_i \in \bar{y}$  **pick a fresh labeled null**  $n_i \in \bar{n}$ ;

**Add to K all the atoms occurring in**  $\psi(\bar{c}, \bar{n})$ ;

**End For;**

**End For;**

**return K;**

We denote by  $ch(M, D)$  the output  $K$  of the algorithm

## The Naïve Chase – Exercise

Consider the source database  $\mathbf{D} = \{R(a, b), S(b, c)\}$  and the mapping  $\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4\}$ , where:

- $\mathbf{m}_1 : \forall x, y. R(x, y) \rightarrow \exists z. A(x, z) \wedge A(z, y)$
- $\mathbf{m}_2 : \forall x, y. S(x, y) \rightarrow \exists z. B(x, z) \wedge B(z, y)$
- $\mathbf{m}_3 : \forall x, y. R(x, y) \rightarrow C(y)$
- $\mathbf{m}_4 : \forall x, y, z. R(x, y) \wedge S(y, z) \rightarrow A(x, z) \wedge B(x, z)$

Compute  $\mathbf{ch}(\mathbf{M}, \mathbf{D})$

## The Naïve Chase – Solution

Consider the source database  $\mathbf{D} = \{R(a, b), S(b, c)\}$  and the mapping  $\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4\}$ , where:

- $\mathbf{m}_1 : \forall x, y. R(x, y) \rightarrow \exists z. A(x, z) \wedge A(z, y)$
- $\mathbf{m}_2 : \forall x, y. S(x, y) \rightarrow \exists z. B(x, z) \wedge B(z, y)$
- $\mathbf{m}_3 : \forall x, y. R(x, y) \rightarrow C(y)$
- $\mathbf{m}_4 : \forall x, y, z. R(x, y) \wedge S(y, z) \rightarrow A(x, z) \wedge B(x, z)$

Compute  $\mathbf{ch}(\mathbf{M}, \mathbf{D})$

- $\mathbf{K} = \{ A(a, \mathbf{z}_1), A(\mathbf{z}_1, b), B(b, \mathbf{z}_2), B(\mathbf{z}_2, c), C(b), A(a, c), B(a, c) \}$   


|                      |                      |                      |                      |        |           |           |
|----------------------|----------------------|----------------------|----------------------|--------|-----------|-----------|
| $A(a, \mathbf{z}_1)$ | $A(\mathbf{z}_1, b)$ | $B(b, \mathbf{z}_2)$ | $B(\mathbf{z}_2, c)$ | $C(b)$ | $A(a, c)$ | $B(a, c)$ |
| $m_1$                | $m_2$                | $m_3$                | $m_4$                |        |           |           |

# The Naïve Chase Algorithm and Universal Solutions

**Theorem C:** Given  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a GLAV information integration specification and  $D$  is a **source database**, we have that  $\text{ch}(M, D)$  is a **universal solution** of  $J$  w.r.t.  $D$

**Proof (sketch):** The (complex) proof is based on the following intuitions:

1. Every solution needs to materialize all the *triggered* mappings
2. The use of fresh new variables guarantees homomorphisms
3. Not materializing any extra information guarantees that we create information that needs to occur in all the semantics of  $J$  w.r.t.  $D$

# Certain Answers and the Chase Algorithm

**Theorem:** Given  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is a GLAV information integration specification and  $D$  is a **source database**, and CQ  $q$ , we have that  $\text{cert}(q, J, D) = q^\perp(\text{ch}(M, D))$

**Proof:** the statement directly follows from **Theorem U** and **Theorem C**

# Query Answering – Recognition Problem

Let **CQ-GLAV** be the class of **conjunctive GLAV mappings**

**DEF:** UCQ-QA-IIS( $\mathcal{L}$ ) where  $\mathcal{L}$  is a class of mappings is the following problem

- **Input:** An I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  where  $\mathbf{G}$  and  $\mathbf{S}$  are empty theories and  $\mathbf{M}$  is a mapping in  $\mathcal{L}$ , a source database  $\mathbf{D}$ , a UCQ  $q$ , and a tuple of constants  $\bar{c}$
- **Question:** Is  $\bar{c} \in \text{cert}(q, J, \mathbf{D})$  ?

**Thm:** UCQ-QA(CQ-GLAV) is **decidable** (actually polynomial in data complexity)

**Proof:** For a GLAV mapping  $\mathbf{M}$  and database  $\mathbf{D}$ , we can compute a **universal solution**  $\mathbf{K}$  of  $\mathbf{D}$  w.r.t.  $\mathbf{M}$  via the **naïve chase** (in polynomial time w.r.t. the size of  $\mathbf{D}$ ) and we can use the **naïve evaluation** to answer the query  $q$  over  $\mathbf{K}$  (in polynomial time w.r.t. the size of  $\mathbf{D}$ )

# The Chase and SQL

Can we use SQL databases to materialize the result of the naïve chase?

- In general no, if we want to use nulls as variables due to two crucial differences
  - We cannot use normal data values for variables
1. **Definition of the result.** SQL is based on Codd nulls, therefore we cannot have repetition of variables.
  2. **Query Answering.** SQL is based on 3-valued logics while we want to compute naïve evaluation of queries over the result of the chase.

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Query Answering in (G)LAV Virtualization

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. **Query answering over IISs**
9. Ontology-Based Data Management (OBDM)
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# **Query Answering in (G)LAV**

# Logical Formalization of Information Integration Systems – Syntax

An information integration system (IIS) is a triple  $J = \langle G, M, S \rangle$  where:

- **G** is the **global schema**
  - A logical theory over a relational alphabet  $A_G$ .
- **S** is the **source schema**
  - A logical theory over a relational alphabet  $A_S$ .
- **M** is the **mapping between S and G**
  - A logical theory over the relational alphabet  $A_S \cup A_G$ .

## Logical Formalization of I.I.S.s – Semantics

Assume a information integration system  $\langle J, D \rangle$ , where  $J = \langle G, M, S \rangle$  is an information integration specification and  $D$  is a **source database** for  $S$

The semantics of  $\langle J, D \rangle$  is defined as follows:

$$\text{sem}(J, D) = \{ B \mid B \text{ is a } \text{global database} \text{ s.t. } (B, D) \models M \}$$

Assume a query  $q$  over  $A_G$ . The **certain answers** of  $q$  w.r.t.  $J$  and  $D$  are:

$$\text{cert}(q, J, D) = \{ \bar{a} \mid \bar{a} \in q(B) \text{ for each } B \in \text{sem}(J, D) \}$$

# Query Answering – Recognition Problem

- **DEF:** **QA-IIS** is the following decision problem:
  - **Input:** an I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , a FOL query  $q(\bar{x})$ , a source database  $\mathcal{D}$ , a tuple of constants  $\bar{c}$
  - **Question:** Is  $\bar{c} \in \text{cert}(q, J, \mathcal{D})$  ?

# Conjunctive Query (recap)

- Let  $\varphi(x_1, \dots, x_n)$  be a FOL formula with free variables  $x_1, \dots, x_n$
- $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction if it is of the form

$$\exists y_1, \dots, \exists y_n P_1(\bar{z}_1) \wedge \dots \wedge P_k(\bar{z}_k)$$

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Conjunctive Query** if  $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction.

## Union of Conjunctive Queries (recap)

- A formula  $\varphi(x_1, \dots, x_n)$  is Union of Existential Conjunctions if it is of the form

$$\bigvee_i \psi_i(x_1, \dots, x_n)$$

where each  $\psi_i$  is an existential conjunction.

**Observe:** same free variables.

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Union of Conjunctive Queries** if  $\varphi(x_1, \dots, x_n)$  is a Union of Existential Conjunctions.

# Our Assumptions

As made for GAV, we assume:

- That **G** is an empty theory over the alphabet  $A_G$ , i.e., **G** is a database schema without integrity constraints.
- That **S** is an empty theory over the alphabet  $A_S$ , i.e., **S** is a database schema without integrity constraints.
- That **GLAV mappings** are defined via queries that we can actually handle: **Conjunctive Queries**

**From now on, if not otherwise stated, we always consider the assumptions made above to be true moreover, if not otherwise stated, we will assume that the user queries are conjunctive queries**

## Local As View (recap)

LAV mappings describe the **source schema** in terms of the **global schema**

A LAV **mapping assertion** is a pair  $\langle s, q_g \rangle$  where:

- $s$  is a **single predicate** name of arity  $n$  in  $A_S$
- $q_g = \{\bar{x} \mid \varphi_g(\bar{x})\}$  is a FOL **query of arity  $n$  over  $A_G$**

Logical formalization of LAV mapping assertions:

$$\forall \bar{x} . s(\bar{x}) \rightarrow \varphi_g(\bar{x})$$

A LAV mapping is a **finite** set of LAV mapping assertions

# Global and Local As View (recap)

Materialization techniques for **LAV** mapping are similar to those for **GLAV**, so we will look at the more general case of **GLAV**

A **GLAV mapping assertion** is a pair  $\langle \mathbf{q}_s, \mathbf{q}_g \rangle$  where:

- $\mathbf{q}_s = \{\bar{x} \mid \varphi_s(\bar{x})\}$  is a FOL query of arity **n** over  $\mathcal{A}_S$
- $\mathbf{q}_g = \{\bar{x} \mid \varphi_g(\bar{x})\}$  is a FOL query of arity **n** over  $\mathcal{A}_G$

Logical formalization of GLAV mapping assertions:

$$\forall \bar{x}. \mathbf{\varphi}_s(\bar{x}) \rightarrow \mathbf{\varphi}_g(\bar{x})$$

A GLAV mapping is a **finite** set of GLAV mapping assertions.

# Conjunctive GLAV Mapping

A **Conjunctive GLAV mapping assertion** is a pair  $\langle \mathbf{q}_s, \mathbf{q}_g \rangle$  where:

- $\mathbf{q}_s = \{\bar{x} \mid \varphi_s(\bar{x})\}$  is a **CQ** of arity **n** over  $\mathcal{A}_S$
- $\mathbf{q}_g = \{\bar{x} \mid \varphi_g(\bar{x})\}$  is a **CQ** of arity **n** over  $\mathcal{A}_G$

A **Conjunctive GLAV mapping** is a **finite** set of Conjunctive GLAV mapping assertions.

# Query Answering – Recognition Problem

- Let **CQ-GLAV** be the class of **conjunctive GLAV mappings**
- **DEF:** **UCQ-QA-IIS( $\mathcal{L}$ )** where  $\mathcal{L}$  is a class of mappings is the following problem
  - **Input:** An I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  where  $\mathbf{G}$  and  $\mathbf{S}$  are empty theories and  $\mathbf{M}$  is a mapping in  $\mathcal{L}$ , a source database  $\mathbf{D}$ , a UCQ  $\mathbf{q}$ , and a tuple of constants  $\bar{c}$
  - **Question:** Is  $\bar{c} \in \text{cert}(\mathbf{q}, J, \mathbf{D})$  ?
- **Thm:** **UCQ-QA(CQ-GLAV)** is **decidable**
  - Via materialization (*see the slide on GLAV via materialization*)
  - Via virtualization (we will provide a proof via **virtualization**)

# **Homomorphisms (recap)**

# The notion of homomorphism

Let **A** and **B** be two database instances over the same schema

A **homomorphism** from **A** to **B** is a function  $h$  from  $\text{dom}(\mathbf{A})$  to  $\text{dom}(\mathbf{B})$   
( $\text{dom}(A)$  and  $\text{dom}(B)$  can contain both *constants* and *variables\labeled nulls*)  
satisfying the following properties:

1. For every constant  $c \in \text{Const}(\mathbf{A})$ ,  $h(c) = c$
2. For every  $R(\bar{a}) \in \mathbf{A}$ , we have  $R(h(\bar{a})) \in \mathbf{B}$

For a tuple  $\bar{a} = (a_1, \dots, a_n)$  of constants and labeled nulls,  $h(\bar{a}) = (h(a_1), \dots, h(a_n))$ . In other words, condition 2. always ensures  $h(\mathbf{A}) \subseteq \mathbf{B}$  whenever  $h$  is a **homomorphism** from **A** to **B**, where  $h(\mathbf{A}) = \{R(h(\bar{a})) \mid R(\bar{a}) \in \mathbf{A}\}$

# Composition of Homomorphisms

Let  $h$  be an homomorphism from  $A$  to  $B$ , and let  $h'$  be an homomorphism from  $B$  to  $C$ . The **composition**  $h' \circ h$  of  $h, h'$  is the mapping defined as follows:  $h' \circ h(x) = h'(h(x))$

**Thm:** The composition  $h' \circ h$  is itself a homomorphism  $h''$  from  $A$  to  $C$ .

**Proof:** Let  $\alpha \in A$ . Then,  $h(\alpha) \in B$  and therefore  $h'(h(\alpha)) \in C$ . The claim follows straightforwardly.

# Finding Homomorphisms

Finding **homomorphisms** is harder than it may seem

- **Thm:** Given two sets of atoms A and B, checking whether there exists a homomorphism from A to B is NP-Complete
- **Thm:** Let A be a set of atoms. For an input set of atoms B, checking whether there exists a homomorphism from A to B is in LOGSPACE
  - Indeed, if we fix A, the number of possible assignments we need to test becomes small

# Query Answering and Homomorphisms (Complete Databases)

We have following well known property:

Let  $q = \{\bar{x} \mid \bigvee_i \psi_i(x_1, \dots, x_n)\}$  be a UCQ and  $D$  be a database instance

**Thm:** a tuple  $\bar{a} \in q(D)$  if and only if there exists  $i = 1, \dots, n$  and an **homomorphism**  $h$  from  $\psi_i$  to  $D$  such that  $h(\bar{x}) = \bar{a}$

# Query Answering In Information Integration

|     | Materialized | Virtual |
|-----|--------------|---------|
| GAV | 1            | 2       |
| LAV | 3            | 4       |

## Virtualization in LAV

We want to produce a query over the **sources** that captures the results of a user query over the **global schema**.

Desired properties of the **source query**.

- We want to use it to compute **certain answers**.
- It should fall into a specific query language.

We cannot use simple unfolding techniques with LAV mappings (as for GAV), since LAV does not define the global schema atoms directly

Indeed, computing a rewriting for LAV is a complex task

# Perfect Rewritings

- Assume an Information Integration System  $J = \langle \textcolor{blue}{G}, \textcolor{green}{M}, \textcolor{brown}{S} \rangle$ .
  - No further assumptions on the three components for now
- Assume a class of queries  $L$  (a language of queries), e.g., CQs, UCQs, FOL, ...
- **Definition:** Given a query  $\textcolor{blue}{q}$  over  $\textcolor{blue}{G}$ , a **perfect  $L$ -rewriting of  $\textcolor{blue}{q}$  w.r.t.  $J$**  is a query  $\textcolor{blue}{rew}_L^{(\textcolor{blue}{q}, J)}$  in the language  $L$  such that the following condition holds for every database  $\textcolor{brown}{D}$  for  $\textcolor{brown}{S}$

$$\bar{a} \in \text{cert}(\textcolor{blue}{q}, J, \textcolor{brown}{D}) \text{ if and only if } \bar{a} \in \textcolor{blue}{rew}_L^{(\textcolor{blue}{q}, J)}(\textcolor{brown}{D})$$

## Perfect Rewritings – Example

Consider an information integration specification  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , where  $\mathbf{M}$  is defined as follows:

$$\mathbf{M} = \{\forall x, y. \textcolor{brown}{TStudents}(x, y) \rightarrow \exists z. \textcolor{blue}{Person}(x) \wedge \textcolor{blue}{Code}(x, z) \wedge \textcolor{blue}{Name}(x, y)\}$$

Consider the following query  $q(\bar{x})$  over  $\mathbf{G}$

$$q = \{ (x) \mid \textcolor{blue}{Person}(x) \}$$

The following query  $q'$  over  $\mathbf{S}$  is a perfect CQ-rewriting of  $q$

$$q' = \{ (x) \mid \exists y. \textcolor{brown}{TStudents}(x, y) \}$$

# Exercise

Consider an information integration specification  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , where  $\mathbf{M}$  is defined as follows:

$$\mathbf{M} = \{\forall x, y. \textcolor{brown}{TStudents}(x, y) \rightarrow \exists z. \textcolor{blue}{Person}(x) \wedge \textcolor{blue}{Code}(x, z) \wedge \textcolor{blue}{Name}(x, y)\}$$

**What is the perfect rewriting of the following queries?**

- $q_1(x) = \{ (x) \mid \exists y. \textcolor{blue}{Code}(x, y) \}$
- $q_2(x, y) = \{ (x, y) \mid \textcolor{blue}{Code}(x, y) \}$
- $q_3(x, y) = \{ (x, y) \mid \textcolor{blue}{Name}(x, y) \}$

## Exercise – Solution

Consider an information integration specification  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$ , where  $\mathbf{M}$  is defined as follows:

$$\mathbf{M} = \{\forall x, y. TStudents(x, y) \rightarrow \exists z. Person(x) \wedge Code(x, z) \wedge Name(x, y)\}$$

What is the perfect rewriting of the following queries?

- $q_1(x) = \{ (x) \mid \exists y. Code(x, y) \}$   
 $q'_1(x) = \{ (x) \mid \exists y. TStudents(x, y) \}$
- $q_2(x, y) = \{ (x, y) \mid Code(x, y) \}$   
 $q'_2() = \{ () \mid \perp \} \text{ (The query always false)}$
- $q_3(x, y) = \{ (x, y) \mid Name(x, y) \}$   
 $q'_3(x, y) = \{ (x, y) \mid TStudents(x, y) \}$

# Perfect Rewritings – Problems

When does a perfect rewriting exists for information integration specifications based on **LAV mappings**?

- The definition is not constructive, and it does not guarantee existence

To study perfect rewritings, we need to fix two languages.

1. The language of **global schema queries** we want to rewrite
2. The language of **source schema queries** we want to use for the rewriting

After fixing a language  $L_G$  of **global schema queries** and a language  $L_S$  of **source schema queries**, we need to face two problems:

1. Is there a perfect  $L_S$ -rewriting **for every** query in  $L_G$ ?
2. Is such rewriting computable?

# Perfect Rewritings in LAV

Fix **UCQ** as **global query language ( $L_G$ )** and **source query language ( $L_S$ )**.

**Is there a perfect UCQ-rewriting for every UCQ in the context of LAV information integration systems?**

We can show that such rewriting always exists and is computable, and we can provide an algorithm for its computation.

The algorithm is based on the following intuitions:

- **Sound Rewritings**: Rewritings that capture only part of the certain answers
- **Maximally-Sound Rewritings**: Rewritings that capture as much as possible in a certain class
- **Query Containment**: Checking whether the answers for a query are always contained in the answers for another.

# Sound Rewritings in LAV

Assume an information integration specification  $J = \langle G, M, S \rangle$ , a query  $q$ , and a class of queries  $L$

- **Definition:** A **sound  $L$ -rewriting of  $q$  w.r.t.  $M$**  is a query  $q'$  in  $L$  such that, for every database  $D$ , the following condition holds:

$$\textit{if } \bar{a} \in q'(D) \textit{ then } \bar{a} \in \textit{cert}(q, J, D)$$

Intuitively, sound rewritings are an approximation of perfect rewritings

**Observe:** such approximations come with no guarantees, indeed, the unsatisfiable query  $\{( \ ) \mid \perp\}$  is always a **sound rewriting**

## Sound Rewritings in LAV – Example

Consider the query  $q(\bar{x}) = \{ x \mid \exists y. A(x, y) \}$

and a mapping  $\mathbf{M} = \{ m_1, m_2 \}$  such that:

- $m_1: \forall x. R(x) \rightarrow \exists y, z. A(x, y) \wedge B(y, z)$
- $m_2: \forall x. S(x) \rightarrow \exists y, z. A(x, y) \wedge C(y, z)$

The following are sound CQ-rewritings of  $q(\bar{x})$

- $q_1 = \{ x \mid R(x) \}$
- $q_2 = \{ x \mid S(x) \}$
- $q_3 = \{ x \mid R(x) \wedge S(x) \}$

# Maximally-Sound Rewritings in LAV

Assume an information integration specification  $J = \langle G, M, S \rangle$ , a query  $q$ , and a class of queries  $L$

- **Definition:** an sound  $L$ -rewriting  $q'$  of  $q$  w.r.t.  $J$  is a **maximally-sound  $L$ -rewriting of  $q$  w.r.t.  $J$**  if for every database  $D$  for  $S$  there exists no sound  $L$ -rewriting  $q''$  of  $q$  w.r.t.  $J$  such that

$$q'(D) \subseteq q''(D)$$

Intuitively, maximally-sound rewritings are one of the **best possible approximations** of certain answers that one can obtain in a language. Of course, they may not be **unique**

## Maximally-Sound Rewritings in LAV – Example

Consider the query  $q(\bar{x}) = \{ x \mid \exists y. A(x, y) \}$

and a mapping  $\mathbf{M} = \{ m_1, m_2 \}$  such that:

- $m_1: \forall x. R(x) \rightarrow \exists y, z. A(x, y) \wedge B(y, z)$
- $m_2: \forall x. S(x) \rightarrow \exists y, z. A(x, y) \wedge C(y, z)$

The following are **maximally-sound CQ-rewritings** of  $q(\bar{x})$

- $q_1 = \{ x \mid R(x) \}$
- $q_2 = \{ x \mid S(x) \}$

The following sound CQ-rewriting of  $q(\bar{x})$  is **not maximal**.

- $q_3 = \{ x \mid R(x) \wedge S(x) \}$

# Computing Perfect UCQ-Rewritings

Consider an information integration specification  $J = \langle G, M, S \rangle$  with  $M$  a LAV mapping, and a UCQ  $q$  over  $G$ .

- **Definition:** Let  $q_m$  be the UCQ defined as the **union of all** the maximally-sound CQ rewritings of  $q$  w.r.t.  $J$

Then we have the following result.

- **Thm:**  $q_m$  is a **perfect UCQ-rewriting** of  $q$  w.r.t.  $J$

# Computing Perfect UCQ-Rewritings – Proof

**Thm:**  $q_m$  is a **perfect UCQ-rewriting** of  $q$  w.r.t.  $J$

**Proof:** We need to prove that  $\bar{a} \in cert(q, J, D)$  if and only if  $\bar{a} \in q_m(D)$ .

Assume  $q_m = \{\bar{x} \mid \bigvee_i \phi_i(\bar{x})\}$ .

**If part.** We need to prove: **If**  $\bar{a} \in q_m(D)$  **then**  $\bar{a} \in cert(q, J, D)$

- Suppose  $\bar{a} \in q_m(D)$ . So, there exists a query  $q_i \in q_m$  such that  $\bar{a} \in q_i(D)$ . By construction, we know that  $q_i$  is a sound rewriting of  $q$ , which means that  $\bar{a} \in cert(q, J, D)$ .

**Only if part.** We need to prove: **If**  $\bar{a} \in cert(q, J, D)$  **then**  $\bar{a} \in q_m(D)$

- This part of the proof is more complex. An intuition follows. (full proof in [S. Abitebul & O.M. Duschka. PODS 1998]. See Proof of Theorem 4.2.)  
If  $\bar{a} \in cert(q, J, D)$  then there exists a CQ  $q_i$  over  $S$  such that

1.  $\bar{a} \in q_i(D)$ , and
2.  $q_i$  is a **sound CQ-rewriting** of  $q$  w.r.t.  $J$ .

Since, by construction,  $q_i$  is a sound rewriting, then  $q_i$  is a disjunct in  $q_m$ . So,  $\bar{a} \in q_m(D)$

# Checking Sound Rewritings

From the theorem above, in order to build a **perfect rewriting** we can:

1. Compute **all maximally sound rewritings** of the given query
2. Output the **union** of all these rewritings

**This is still not an algorithm!**

1. **How can we generate the maximally sound rewritings?**
2. **How many of them are there?**

To answer these questions we must first introduce the following notions:

- **Query containment**
- **Query Expansion**

# Expansion

Consider an information integration specification  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  with  $\mathbf{M}$  a LAV mapping, and a CQ  $q_S$  over  $\mathbf{S}$ .

**Def:** The **expansion**  $\exp^{(q_S, M)}$  of  $q_S$  w.r.t.  $M$  is the query over  $\mathbf{G}$  obtained by replacing each atom  $R(\bar{z})$  in  $q_S$  with the view  $q_R$  over  $\mathbf{G}$  such that  $\langle R, q_R \rangle \in M$

**Observe:**  $\exp^{(q_S, M)}$  is a conjunctive query!

Intuitively, we use  $M$  as a **reverse mapping** from the **schema** to the **global schema** and unfold  $q_S$  over  $M$  in a GAV fashion.

## Expansion – Example

Assume the following queries over a source schema  $\mathbf{S}$ .

- $q_1 = \{ x \mid R(x) \}$
- $q_2 = \{ x \mid R(x) \wedge S(x) \}$

Assume the mapping  $\mathbf{M}$  containing the following assertions.

- $m_1 : \forall x. R(x) \rightarrow \exists y, z. A(x, y) \wedge B(y, z)$
- $m_2 : \forall x. S(x) \rightarrow \exists y, z. A(x, y) \wedge C(y, z)$

The following are the expansions of  $q_1$  and  $q_2$  w.r.t.  $\mathbf{M}$ .

- $\text{exp}^{(q_1, M)} = \{ x \mid \exists y, z. A(x, y) \wedge B(y, z) \}$
- $\text{exp}^{(q_2, M)} = \{ x \mid \exists y, z, v, w. A(x, y) \wedge B(y, z) \wedge A(x, v) \wedge C(v, w) \}$

# Query Containment

**Definition:** Let  $q$  and  $q'$  be two query of the same arity over a schema  $S$ . **Query containment** is the problem of checking whether  $q(D) \subseteq q'(D)$ , for every database  $D$  for  $S$ . We write  $q \sqsubseteq q'$  to denote that  $q$  is contained in  $q'$ .

**Thm:** Let  $q \in q'$  be two CQs query of the same arity over  $S$ . We have that  $q \sqsubseteq q'$  if and only if there exists a **homomorphism**  $h$  from  $q'$  to  $q$  such that  $h(\bar{x}_1) = \bar{x}_2$

# Checking Sound Rewritings

Consider an information integration specification  $J = \langle G, M, S \rangle$  with  $M$  a LAV mapping, a CQ  $q_S$  over  $S$  and CQ  $q_G$  over the global schema  $G$ .

We can use expansion to check sound rewritings:

Thm:  $q_S$  is a **sound rewriting** of  $q_G$  if and only if  $\exp^{(q_S, M)} \sqsubseteq q_G$ , i.e.  
 $\exp^{(q_S, M)}(D) \subseteq q_G(D)$ , for every database  $D$  for  $G$

Proof: The (complex) proofs uses the fact that  $\exp^{(q_S, M)}$  mimics the chase

# Size of Maximally-Sound CQ-Rewritings in LAV

Consider an information integration specification  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  with  $\mathbf{M}$  a LAV mapping, and a CQ  $q_G$  over  $\mathbf{G}$ . The previous **Theorem** provides a way to check if a query  $q_S$  over  $\mathbf{S}$  is a **sound CQ-rewriting** of  $q_G$ .

**What about maximally-sound CQ-rewritings?**

We have the following:

- **Thm:** Let  $q_G = \{\bar{x} \mid \varphi(\bar{x})\}$  be a CQ over  $\mathbf{G}$  such that  $\varphi$  consists of **n distinct atoms**. If a CQ  $q_S = \{\bar{x} \mid \psi(\bar{x})\}$  is a **maximally-sound CQ-rewriting of  $q_G$**  w.r.t.  $J$ , then  $\psi$  is logically equivalent to an existential conjunction of **at most n distinct atoms**.

This provides us with an algorithm for computing the set of maximal conjunctive rewritings 

# The Perfect Rewriting Algorithm

**Input:** An I.I.S.  $J = \langle G, M, S \rangle$  where  $M$  is a Conjunctive LAV mapping, and a CQ  $q_g = \{ \bar{x} \mid \varphi(\bar{x}) \}$  where  $\varphi$  has  $n$  conjuncts.

**Output:** A UCQ  $Q$  over  $S$

$Q \leftarrow \emptyset$

For each CQ  $q' = \{ \bar{x} \mid \psi(\bar{x}) \}$  over  $S$  with at most  $n$  conjuncts

If  $q'$  is a sound rewriting of  $q_g$ , i.e., if  $\exp^{(q', M)} \sqsubseteq q_g$

$Q \leftarrow Q \cup \{q'\}$

End If;

End For;

return  $Q$ ;

# Perfect UCQ-Rewriting in LAV

- **Thm:** Let  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  be an LAV information integration specification and  $q_{\mathcal{G}}$  be a CQ over  $\mathbf{G}$ . For input  $\mathbf{J}$  and  $q_{\mathcal{G}}$ , the **Perfect Rewriting algorithm** computes a perfect UCQ-rewriting of  $q_{\mathcal{G}}$  w.r.t.  $\mathbf{J}$ .
- **Proof:** The algorithm computes the union of all the maximally-sound CQ-rewritings of the input query.
- **Thm:** The algorithm runs in EXPTIME w.r.t. the size of  $q_{\mathcal{G}}$ .

## Dealing with GLAV

We can think of GLAV mappings as the composition of a LAV and a GAV mapping. Indeed, each GLAV mapping  $\langle q_S, q_G \rangle$  can be rewritten as

$$\langle q_S, p \rangle \cup \langle p, q_G \rangle$$

In logical terms we split the implications into two different assertions:

$$\text{GLAV: } \forall \bar{x}. \varphi_S(\bar{x}) \rightarrow \varphi_G(\bar{x})$$



$$\text{GAV: } \forall \bar{x}. \varphi_S(\bar{x}) \rightarrow p(\bar{x});$$

$$\text{LAV: } \forall \bar{x}. p(\bar{x}) \rightarrow \varphi_G(\bar{x})$$

# Dealing with GLAV

Assume the GLAV mapping  $M$  containing the following assertions.

- $m_1 : \forall x. \exists w. R(x) \wedge T(x, w) \rightarrow \exists y, z. A(x, y) \wedge B(y, z)$
- $m_2 : \forall x. \exists w. S(x) \wedge K(w, x) \rightarrow \exists y, z. A(x, y) \wedge C(y, z)$

The following LAV and GAV mappings **together** are equivalent to  $M$

## - GAV

- $m_1^G : \forall x. \exists w. R(x) \wedge T(x, w) \rightarrow G_1(x)$
- $m_2^G : \forall x. \exists w. S(x) \wedge K(w, x) \rightarrow G_2(x)$

## - LAV

- $m_1^L : \forall x. G_1(x) \rightarrow \exists y, z. A(x, y) \wedge B(y, z)$
- $m_2^L : \forall x. G_2(x) \rightarrow \exists y, z. A(x, y) \wedge C(y, z)$

# Perfect UCQ-Rewriting in GLAV

To deal with GLAV in a virtualized way, we can proceed in three steps.

Consider an information integration specification  $\mathbf{J} = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  with  $\mathbf{M}$  a GLAV mapping, and a CQ  $q_{\mathbf{G}}$  over  $\mathbf{G}$

1. From  $\mathbf{M}$ , we define an equivalent pair of mappings  $\mathbf{M}_1$  and  $\mathbf{M}_2$  where  $\mathbf{M}_1$  is LAV and  $\mathbf{M}_2$  is GAV.
  2. We rewrite  $q$  w.r.t.  $\mathbf{M}_1$  using the perfect rewriting algorithm for LAV defined above. Let  $q'$  be the result of this step.
  3. We unfold  $q'$  w.r.t.  $\mathbf{M}_2$  to obtain a query  $q''$  over the schema  $\mathbf{S}$  (as in GAV)
- **Thm:**  $q''$  is a **perfect UCQ-rewriting** of  $q$  w.r.t.  $\mathbf{J}$ .

# Query Answering – Recognition Problem

Let **CQ-GLAV** be the class of **conjunctive GLAV mappings**

**DEF:** **UCQ-QA-IIS( $\mathcal{L}$ )** where  $\mathcal{L}$  is a class of mappings is the following problem

- **Input:** An I.I.S.  $J = \langle \mathbf{G}, \mathbf{M}, \mathbf{S} \rangle$  where  $\mathbf{G}$  and  $\mathbf{S}$  are empty theories and  $\mathbf{M}$  is a mapping in  $\mathcal{L}$ , a source database  $\mathbf{D}$ , a UCQ  $q$ , and a tuple of constants  $\bar{c}$
- **Question:** Is  $\bar{c} \in \text{cert}(q, J, \mathbf{D})$  ?

**Thm:** **UCQ-QA(CQ-GLAV)** is **decidable** (actually LOGSPACE in data complexity)

**Proof:** For a GLAV mapping  $\mathbf{M}$  a query  $\mathbf{q}$  over  $\mathbf{G}$  and database  $\mathbf{D}$  for  $\mathbf{S}$ , we can compute the **perfect UCQ-rewriting**  $\mathbf{q}'$  of  $\mathbf{q}$  w.r.t.  $\mathbf{J}$  (independently from the size of  $\mathbf{D}$ ) and then we can evaluate  $\mathbf{q}'$  over  $\mathbf{D}$  in polynomial time w.r.t. the size of  $\mathbf{D}$  (actually in LOGSPACE).

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# **Advanced Data Management**

## **Ontology-based Data Management**

**Domenico Fabio Savo**

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. **Ontology-Based Data Management (OBDM)**
10. Description Logic Ontologies
11. Query answering in ontologies
12. Query answering in OBDM

# The Problem

## Data management: a complex task

Extracting information and obtaining insights from data, as well as data governance are both challenging but necessary tasks

- the amount of data stored in current information systems continues to grow.
- proliferation of **heterogenous** and independent data sources both within a single organization and in cooperating environments.

We can easily understand why effectively accessing, **integrating**, and managing data in complex organizations is still **one of the main issues faced by the information technology (IT) today**

# Data Integration

Databases are great!

They let us manage efficiently huge amounts of data . . .

. . . assuming you have put all data into your schema.

Unfortunately, the reality is much more complicated and **heterogeneous**:

- Data sets were created independently.
- Data are often stored across different sources.
- Data sources are controlled by different people/organizations.

**Goal of data integration:** put together **different data sources**,

- created for **different purposes**,
- and controlled by **different people**,
- making them **accessible in a uniform way**.

# Physical vs Conceptual Heterogeneity

- **Physical Heterogeneity:** Information is represented using different formats. Datatypes and data models of the information sources may differ.
- **Conceptual Heterogeneity:** The information sources describe the world from different perspectives.

# Is Heterogeneity the only problem?

Gathering information can be **difficult** even for **very simple information systems**

**EX:** This table contains information about the customers of a (real) organization, but...

| CUC     | TS_START    | TS_END      | ID_GRUP | FLAG_CP | FLAG_CF | FATTURATO | FLAG_FATT |
|---------|-------------|-------------|---------|---------|---------|-----------|-----------|
| 124589  | 30-lug-2004 | 1-gen-9999  | 92736   | S       | N       | 195000,00 | N         |
| 140904  | 15-mag-2001 | 15-giu-2005 | 35060   | N       | N       | 230600,00 | N         |
| 124589  | 5-mag-2001  | 30-lug-2004 | 92736   | N       | S       | 195000,00 | S         |
| -452901 | 13-mag-2001 | 27-lug-2004 | 92770   | S       | N       | 392000,00 | N         |
| 129008  | 10-mag-2001 | 1-gen-9999  | 62010   | N       | S       | 247000,00 | S         |
| -472900 | 10-mag-2001 | 1-gen-9999  | 62010   | S       | N       | 0 00      | N         |
| 130976  | 7-mag-2001  | 9-lug-2003  | 75680   |         |         |           |           |

# Is Heterogeneity the only problem?

...contains information about the customers of a (real) organization, but...

- **CUC** column contains the customer code: if it is positive, then it refers to an **ordinary** customer; if it is negative, to a **special** customer.
- **TS\_START** and **TS\_END** specify the time interval of validity for the record.
- **ID\_GROUP** indicates the group the customer belongs to (if the value of **FLAG\_CP** is “S,” then the customer is the **leader** of the **group**; if **FLAG\_CF** is “S,” then the customer is the **controller** of the group).
- **FATTURATO** is the annual turnover (it is valid only if **FLAG\_FATT** is “S”).

Obviously, each term such as **special**, **ordinary**, **leader**, etc. has a specific meaning in the company, and understanding such meaning is crucial for extracting **useful information**

## Data sources evolve in a chaotic way

Although the initial design of data sources in an organization might be adequate, corrective maintenance actions tend to reshape these sources into a form that **diverges from the original structure**.

Data sources are often subject to **changes** to be adapted to new needs and guaranteeing their seamless usage within the organization is costly.

The result is that the data are stored in different sources and tend to be redundant, mutually inconsistent, and **obscure** for large classes of users.

## A new paradigm

The preceding examples and observations show that it is not only important to provide an **unified access** to data, but that it is also critical to know how the data are **organized** and how to **interpret** them (their semantics).

A promising direction for addressing these challenges is provided by the **Ontology-Base Data Integration (OBDI)** paradigm.

# Ontology-based Data Management

**Ontology-based Data Management (OBDM)** is a semantic paradigm, rooted on the idea of using suitable techniques from the area of **knowledge representation and reasoning** in AI for a new way to achieve data governance and integration, based on the principle of managing heterogeneous data through the lens of an **ontology**.

It is characterized by the following principles:

- Data may reside where they are (virtual approach)
- Build a conceptual specification of the domain of interest, in terms of knowledge structures (the ontology)
- Map such knowledge structures to concrete data sources
- Express the user queries over the knowledge structures
- Automatically translate the queries over the data sources

# What is an ontology?

We can define an **ontology** as an explicit specification of a conceptualization, i.e., an **abstract, simplified** view of the world that we wish to represent.

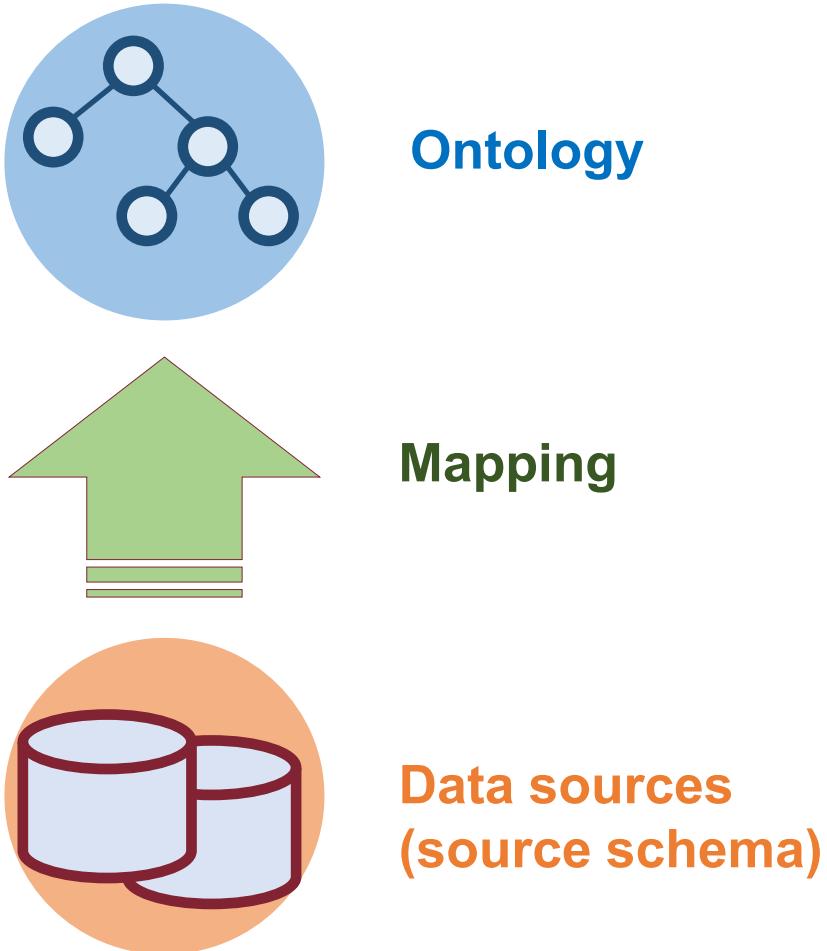
In other words, an ontology is a description of the **concepts** and **relationships** that can exist in a specific domain of interest for an agent or a community of agents.

In AI, an **ontology** provides a set of definitions that associate a name to the entities in a domain (e.g., classes, relations, individuals, etc.), and **formal axioms** that describe the relations between such entities.

Formally, an ontology is a **logical theory**.

The most common languages used for expressing ontologies are **Description Logics** and **OWL** (Web Ontology Language), the W3C standard language for ontologies.

# Ontology-based Data Management: architecture



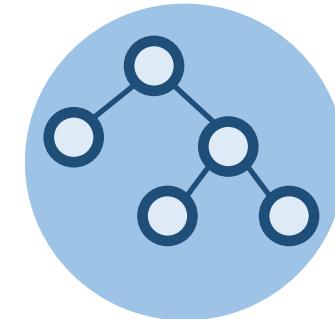
Based on three main components:

- The **Ontology** provides a formal conceptualization of a domain of interest
- The **Data sources**, representing external, independent, heterogeneous, data structures
- The **Mappings**, specifies the relationship between **ontology** elements and data at the **sources**

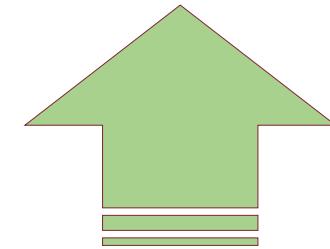
Users access only the conceptual layer, while the data layer, hidden to the clients, manages the data and remain autonomous

# OBDM and Information Integration

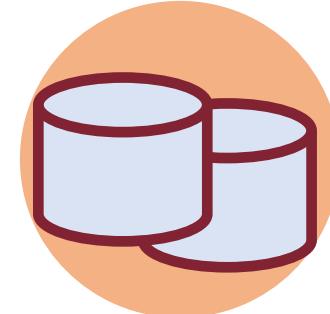
OBDM can be seen as a sophisticated form of **information integration** where the usual **global schema** is replaced by the conceptual model of the application domain, formulated as an **ontology**.



Ontology



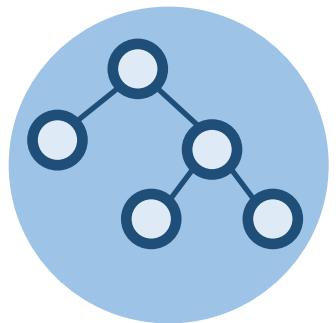
Mapping



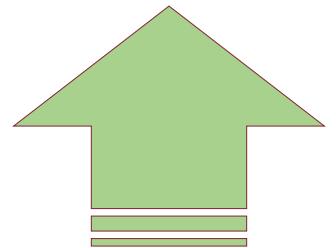
Data sources  
(source schema)

The integrated view is not anymore merely a data structure accommodating the various data at the **sources**, but a **semantically rich description** of the concepts of the domain and the relationships between them.

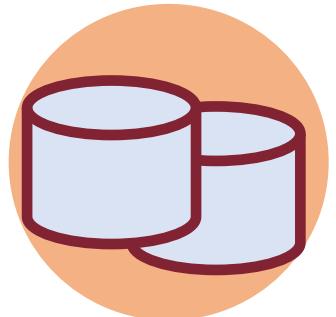
# OBDM vs Data Integration



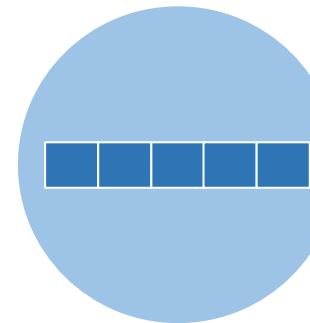
Ontology



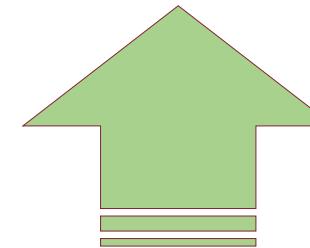
Mapping



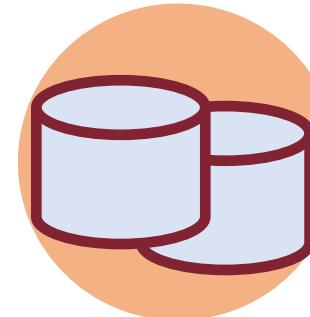
Data sources  
(source schema)



Global Schema



Mapping



Data sources  
(source schema)

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Ontologies and Knowledge Engineering

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. Ontology-Based Data Management (OBDM)
10. Description Logic Ontologies
- 11. Knowledge Engineering**
12. Query answering in ontologies
13. Query answering in OBDM

# Introduction to Ontologies

# What is an Ontology?

**Definition:** An **ontology** is a formal, explicit specification of a shared conceptualization.

## Key aspects:

- **Formal:** machine-readable
- **Explicit:** concepts, properties, relations, functions, constraints, and axioms are explicitly defined
- **Shared:** consensual knowledge
- **Conceptualization:** abstract model of phenomena in the world

Origins in philosophy, adapted for computer science and AI

# Ontologies in AI and Knowledge Representation

Ontologies help in:

- **Knowledge Sharing:** Enable communication between systems.
- **Knowledge Reuse:** Reusable in different contexts.
- **Automated Reasoning:** Using logical rules to infer new knowledge.

**Example:** Reuse of ontologies in e-commerce and healthcare.

# Why Ontologies?

- Enable **knowledge sharing**
- Provide a **common understanding** of a domain
- Make domain assumptions **explicit**
- Separate **domain knowledge** from operational knowledge
- Enable **reuse** of domain knowledge
- Facilitate **interoperability** between systems
- Automated **Reasoning**, using logical rules to infer new knowledge.

# Components of Ontologies

- **Concepts:** represent sets of objects
- **Roles:** ways in which classes and instances can be related to one another
- **Instances:** specific membership of objects in classes and relationships
- **Attributes:** relationships between concepts and concrete domains
- **Axioms:** assertions in a logical form about the classes, instances, and relations

# Components Example

**Domain:** University

- **Classes:** Person, Student, Professor, Course
- **Roles:** teaches, enrolledIn
- **Instances:** Student(John), Course(CS), enrolledIn (John, CS)
- **Properties:** name, courseCode
- **Axioms:** Professor  $\sqsubseteq$  Person, Student  $\sqsubseteq \exists$  enrolledIn

# Types of Ontologies

- **Upper (Top-level) Ontologies**: describe very general concepts applicable across domains. Examples: SUMO, BFO, DOLCE
- **Domain Ontologies**: represent concepts in a specific domain (e.g., medicine, finance)
- **Task Ontologies**: describe vocabulary for a specific task or activity (e.g., ontologies for web search)
- **Application Ontologies**: combine domain and task ontologies for a specific application

# Ontology Languages

## Mathematical (logical) Languages:

- FOL
- Description Logics
- Datalog
- ...

## Concrete Languages (machine readable):

- RDF (Resource Description Framework): subject-predicate-object triples
- RDFS (RDF Schema): extends RDF with basic ontological modeling.
- OWL 2 (Web Ontology Language): based on Description Logics, equipped with profiles (OWL-Lite, OWL-DL, OWL-Full), W3C standard.

# Example of a Simple Ontology in DL

## TBox:

$\delta(\text{name}) \sqsubseteq \text{Person}$   
 $\rho(\text{name}) \sqsubseteq \text{xsd:string}$   
 $\text{Professor} \sqsubseteq \text{Person}$   
 $\text{Student} \sqsubseteq \text{Person}$   
 $\text{Student} \sqsubseteq \exists \text{enrolledIn}$

## ABox:

$\text{Course(CS)}$   
 $\text{Student(john)}$   
 $\text{enrolledIn(john, CS)}$

# Example of a Simple Ontology in OWL

```
Declaration(Class(Course))
Declaration(Class(Person))
Declaration(Class(Professor))
Declaration(Class(Student))
Declaration(ObjectProperty(enrolledIn))
Declaration(ObjectProperty(teaches))
Declaration(DataProperty(name))
Declaration(NamedIndividual(CS))
Declaration(NamedIndividual(john))

DataPropertyDomain(name Person)
DataPropertyRange(name xsd:string)

SubClassOf(Professor Person)
SubClassOf(Student Person)
SubClassOf(Student ObjectSomeValuesFrom(enrolledIn owl:Thing))

ClassAssertion(Course CS)

ClassAssertion(Student john)
ObjectPropertyAssertion(enrolledIn john CS)
```

# Knowledge Engineering

# What is Knowledge Engineering?

**Definition:** Process of designing, building, and managing knowledge-based systems.

## Key Activities:

- Knowledge acquisition
- Knowledge modeling
- Knowledge validation and maintenance

# Knowledge Engineering Cycle

## Key steps:

1. **Domain analysis and Knowledge Acquisition**: gathering knowledge from experts and literature.
2. **Conceptualization**: structuring knowledge into models.
3. **Ontology development**: using formal languages like OWL or RDF.
4. **Ontology population**: specifying instances of concepts, roles and attributes
5. **Ontology evaluation and refinement**: testing the accuracy and efficiency of the ontology.
6. **Ontology maintenance and evolution**: Keeping ontologies up to date.

# 1 - Domain Analysis and Knowledge Acquisition

The first step requires to identify the domain and scope of the ontology

## Gather domain knowledge from:

- Domain experts
- Existing ontologies
- Textbooks and scientific literature
- Databases and other structured sources

## Techniques:

- Interviews with domain experts
- Document analysis
- Collaborative workshops

## 2 - Conceptualization

- Organize and structure the acquired knowledge
- Identify key **concepts**, **relations**, and **properties**
- Create **informal** representations:
  - Concept maps
  - UML diagrams
  - Glossaries and taxonomies
- **Validate** conceptualization with domain experts

## 3 - Ontology Design and Development

- Choose an appropriate **ontology language** (e.g., OWL)
- Formalize the conceptualization:
  - Define **classes** and class **hierarchies**
  - Define **properties** and **relations**
  - Create **instances**
  - Specify **axioms** and rules
- Use ontology development tools (e.g., **Protégé**, Eddy, etc. )

## 4 - Ontology Population

- Add **instances** (individuals) to the ontology
- Methods:
  - Manual population by domain experts
  - Semi-automatic population using:
    - Natural Language Processing techniques
    - Machine Learning algorithms
  - Automatic population from structured data sources (e.g. OBDM)
- Ensure **consistency** and **quality** of added instances

## 5 - Ontology Evaluation and Refinement

- Verify the ontology meets its **requirements** and competency questions
- **Evaluation criteria:**
  - Consistency
  - Completeness
  - Conciseness
  - Expandability
- **Techniques:**
  - Automated reasoning
  - Expert review
  - Application-based evaluation

## 6 - Ontology Maintenance and Evolution

Ontologies are dynamic and need to evolve

- Reasons for evolution:
  - Changes in the domain
  - New requirements
  - Errors or inconsistencies discovered
- Versioning and change management
- Collaborative ontology development and maintenance

# Approaches for Building Ontologies

- **Manual Engineering:** ontologies developed by experts.
- **Semi-automatic:** tools assist in suggesting classes/relations.
- **Fully Automatic:** algorithms extract knowledge (not common due to complexity).

# Popular Ontology Development Methodologies

Over the years, various methodologies for developing structured ontologies have been proposed.

Some of these are:

- **Methontology**: A structured method including stages of specification, conceptualization, integration, and evaluation.
- **Ontology Development 101**: Simpler approach focusing on identifying classes, hierarchy, and properties.
- **NeON Methodology**: Focuses on reuse and networking of ontologies (integration of existing ontologies).

# Practical Applications of Ontologies

- Semantic Web and Linked Data
- Knowledge Management Systems
- Information Retrieval and Question Answering
- Natural Language Processing
- Data Integration and Interoperability
- Decision Support Systems
- Bioinformatics and Healthcare
- E-commerce and Recommender Systems

# Applications and Case Studies

## Case Study 1: Biomedical Ontologies

- Ontologies like **SNOMED CT, Gene Ontology, and HPO** use DL for formalizing medical knowledge.
- **Example:** Defining hierarchical relationships between diseases and symptoms.

## Case Study 2: Ontologies in the Semantic Web

- **FOAF (Friend of a Friend):** An ontology to describe people and their social networks.
- **Example:** Using DL to infer indirect social relationships.

# Challenges in Ontology Engineering

- **Scalability:** Large ontologies can become difficult to manage and reason over.
- **Integration:** Merging ontologies from different domains can lead to conflicts.
- **Human Expertise:** Requires significant domain knowledge to develop useful ontologies.

# Advanced Data Management

## Description Logic Ontologies

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. Ontology-Based Data Management (OBDM)
- 10. Description Logic Ontologies**
11. Query answering in ontologies
12. Query answering in OBDM

# **Ontologies for information management**

# Ontologies

**Definition [Ontologies in Computer Science]:** An **ontology** is a representation scheme describing a **formal conceptualization** of a domain of interest.

The specification of an **ontology** usually comprises two distinct levels:

- **Intensional level:** specifies a set of **conceptual elements** and of constraints/axioms describing the conceptual structures of the domain.
- **Extensional level:** specifies a set of **instances** of the conceptual elements described at the intensional level.

# Description Logic Ontologies

The formal foundations for ontology languages are in logic, and specifically in **Description Logics (DLs)**.

**Description logics** are fragments of **first-order logic** specifically tailored towards the representation of structured knowledge.

- Formalisms in logic are used to provide data with a **formal semantics**
- The logic-based formalization allows one to provide **automated support** for tasks related to data management, by means of **logic-based inference**.
- **Computational aspects** are of concern, so that tools can provide effective support for automated reasoning.

**In this course we adopt DLs for expressing ontologies**

# Conceptual Schemas in Information Systems

**Intensional information** has traditionally played an important role in information systems.

**Design phase** of the information system:

- **Conceptual modeling**: from the requirements, a **conceptual schema** of the domain of interest is produced.
  - The **conceptual schema** is used to produce the **logical data schema**.
  - The data are stored according to the logical schema, and queried through it.

**Conceptual Modeling**: The activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication.

# Ontologies in Information Systems

The role of ontologies in information systems goes beyond that of conceptual schemas.

**Ontologies** affect the whole life-cycle of the information system:

- Ontologies, with the associated reasoning capabilities and inference tools, can provide support at design time.
- The use of ontologies can significantly simplify **maintenance** of the information system's data assets.
- The ontology is used also to support the interaction with the information system and to improve communication between organizations
- Ontology-based Data Management

# Issues in ontology-based information management

## Which is the right ontology language?

- many proposals with different expressive power and complexity of inference have been made

## Which languages should we use for querying?

- requirements for querying are different from those for modeling

## How do we connect the ontology to available data sources?

- mismatch between the information in an ontology and data in a data source

## How can we handle large ontologies?

- one needs to consider the **tradeoff** between expressive power and complexity of inference.

## How can we cope with large amounts of data?

- what is good for large ontologies, may not be good enough for large amounts of data.

# Introduction to Description Logics

# What are Description Logics?

**Description Logics** (DLs) are logics specifically designed to represent and reason on structured knowledge.

The domain of interest is composed of objects and is structured into:

- **concepts**, which correspond to classes, and denote sets of objects
- **roles**, which correspond to (binary) relationships, and denote binary relations on objects

The knowledge is asserted through so-called **assertions**, i.e., logical axioms.

# What are Description Logics about?

Abstractly, DLs allow one to predicate about labeled directed graphs:

- Nodes represent real world objects.
- Node labels represent types/categories of objects.
- Edges represent relations between (pairs of) objects, or between objects and values
- Edge labels represent the types of relations between objects, or between objects and values.

Every fragment of the world that can be abstractly represented in terms of a labeled directed graph is a good candidate for being represented by DLs.

# What are Description Logics about?

A **DL** is characterized by:

A **description language**: how to form concepts and roles

- $\text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer})$

A mechanism to **specify knowledge** about concepts and roles (i.e., a **TBox**)

- $T = \{ \text{Father} \equiv \text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild};$   
 $\quad \text{HappyFather} \sqsubseteq \text{Father} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer}) \}$

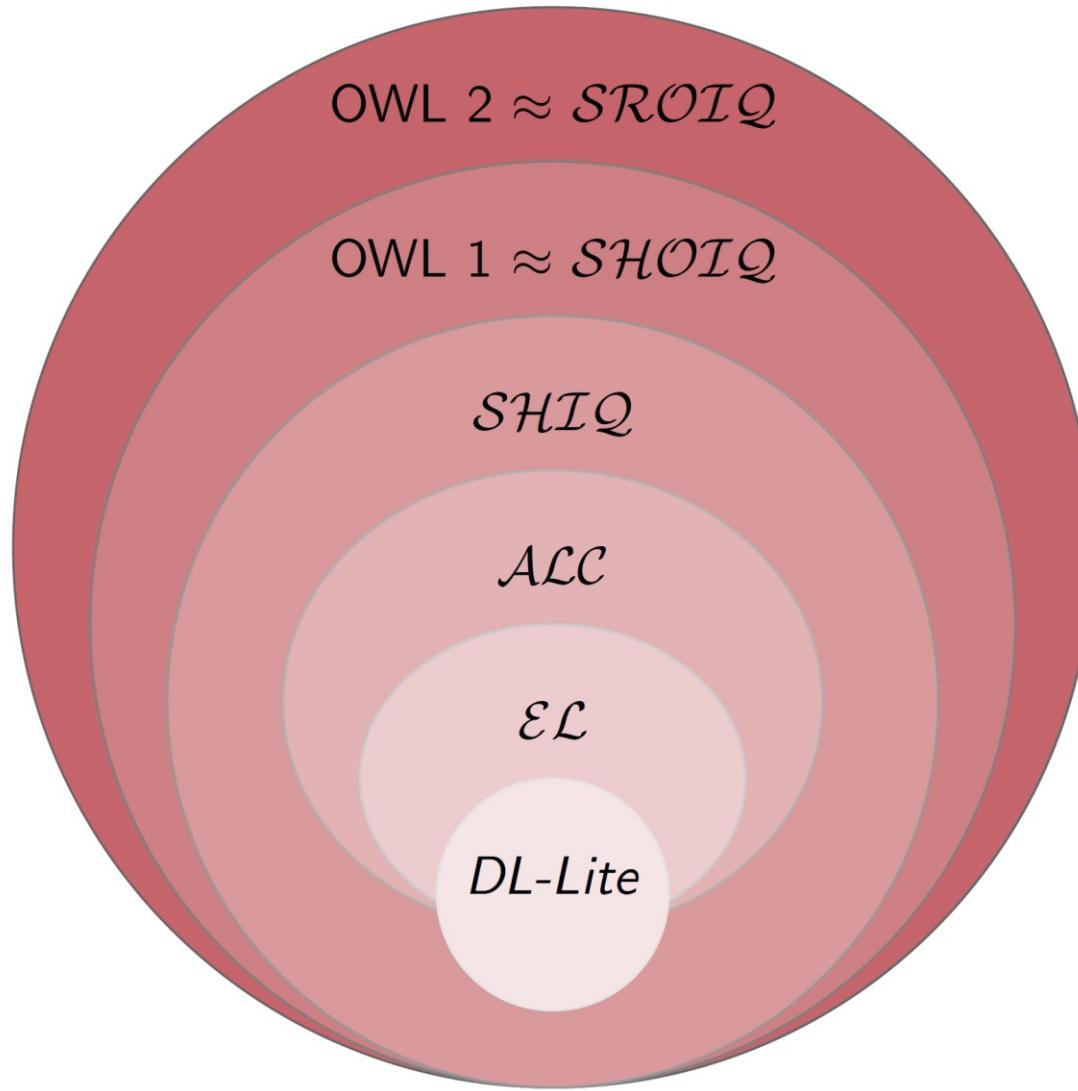
A mechanism to specify **properties of objects** (i.e., an **ABox**)

- $A = \{ \text{HappyFather(john)}; \text{hasChild(john, mary)} \}$

A set of **inference services**: how to reason on a given ontology

- $T \cup A \models (\text{Doctor} \sqcup \text{Lawyer})(\text{mary})$

# Many description logics



# Description Language

A **description language** provides the means for defining:

- **concepts**, corresponding to classes: interpreted as sets of objects;
- **roles**, corresponding to relationships: interpreted as binary relations on objects.

To define concepts and roles:

- We start from a (countably infinite) alphabet of concept names and role names, forming the so called **atomic concepts and roles**.
- Then, by applying specific **constructors**, we can build **complex concepts and roles**, starting from the atomic ones.

A **description language** is characterized by the set of **constructs** that are available for that.

# Formal semantics of a description language

The **formal semantics** of DLs is given in terms of **interpretations**.

**Definition:** An **interpretation**  $I = (\Delta^I, \cdot^I)$  consists of:

- a non-empty set  $\Delta^I$  called the **interpretation domain** (of  $I$ )
- an **interpretation function**  $\cdot^I$ , which maps:
  - each atomic concept  $A$  to a subset  $A^I$  of  $\Delta^I$
  - each atomic role  $P$  to a subset  $P^I$  of  $\Delta^I \times \Delta^I$

The **interpretation** function is extended to complex concepts and roles according to their syntactic structure.

# Concept constructors

| Construct                               | Syntax         | Example                 | Semantics                                                         |
|-----------------------------------------|----------------|-------------------------|-------------------------------------------------------------------|
| <b>Atomic concept</b>                   | $A$            | Student                 | $A^I \subseteq \Delta^I$                                          |
| <b>Atomic role</b>                      | $P$            | LivesIn                 | $P^I \subseteq \Delta^I \times \Delta^I$                          |
| <b>Atomic negation</b>                  | $\neg A$       | $\neg$ Student          | $\Delta^I \setminus A^I$                                          |
| <b>Conjunction</b>                      | $C \sqcap D$   | Student $\sqcap$ Worker | $C^I \cap D^I$                                                    |
| <b>(unqual) existential restriction</b> | $\exists R$    | $\exists$ LivesIn       | $\{ o \mid \exists o'. (o, o') \in R^I \}$                        |
| <b>Value restriction</b>                | $\forall R. C$ | $\forall$ LivesIn.City  | $\{ o \mid \forall o'. (o, o') \in R^I \rightarrow o' \in C^I \}$ |
| <b>Bottom</b>                           | $\perp$        |                         | $\emptyset$                                                       |

(C, D denote arbitrary concepts and R an arbitrary role)

The above constructs form the basic language  $\mathcal{AL}$  of the family of  $\mathcal{AL}$  languages.

# Additional concept and role constructors

| Construct                            | Syntax        | $\mathcal{AL}$ | Semantics                                                   |
|--------------------------------------|---------------|----------------|-------------------------------------------------------------|
| <b>Disjunction</b>                   | $C \sqcup D$  | $u$            | $C^I \cup D^I \subseteq \Delta^I$                           |
| <b>Top</b>                           | $T$           |                | $\Delta^I$                                                  |
| <b>(full) negation</b>               | $\neg C$      | $c$            | $\Delta^I \setminus C^I$                                    |
| <b>Qual. existential restriction</b> | $\exists R.C$ | $\varepsilon$  | $\{ o \mid \exists o'.(o, o') \in R^I \wedge o' \in C^I \}$ |
| <b>Number restriction</b>            | $(\geq k R)$  | $\mathcal{N}$  | $\{ o \mid \#\{o' \mid (o, o') \in R^I\} \geq k \}$         |
|                                      | $(\leq k R)$  |                | $\{ o \mid \#\{o' \mid (o, o') \in R^I\} \leq k \}$         |
| <b>Inverse role</b>                  | $R^-$         | $I$            | $\{(o', o) \mid (o, o') \in R^I\}$                          |
| :                                    | :             | :              | :                                                           |

Many different DL constructs and their combinations have been investigated.

## DL constructs – Example

- Disjunction  $\forall \text{hasChild}. (\text{Doctor} \sqcup \text{Lawyer})$
- Inverse role  $\exists \text{hasChild}^{-}. \text{Doctor}$
- (full) negation  $\neg(\text{Doctor} \sqcup \text{Lawyer})$
- Qualified existential restriction  $\exists \text{hasChild}^{-}. \text{Doctor}$
- Number restriction  $(\geq 2 \text{ hasChild}) \sqcap (\leq 1 \text{ sibling})$

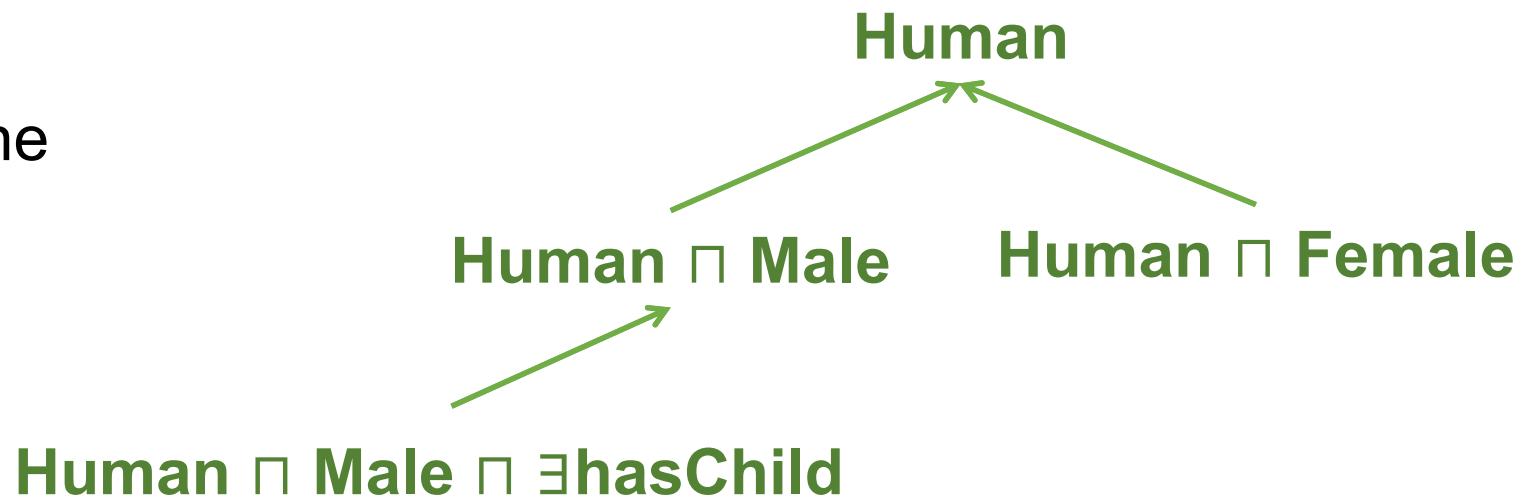
# Reasoning over concept expressions

An **interpretation  $I$**  is a model of a concept  $C$  if  $C^I \neq \emptyset$ .

Basic reasoning tasks:

- Concept **satisfiability**: does  $C$  admit a model?
- Concept **subsumption**:  $C \sqsubseteq D$ : is  $C^I \subseteq D^I$  for every interpretation  $I$ ?

Subsumption is used to build the  
**concept hierarchy**



# Complexity of reasoning over concept expressions

Complexity of concept satisfiability:

|                                                 |                 |
|-------------------------------------------------|-----------------|
| $\mathcal{AL}, \mathcal{ALN}$                   | PTime           |
| $\mathcal{ALU}, \mathcal{ALUN}$                 | NP-Complete     |
| $\mathcal{ALE}$                                 | CoNP-Complete   |
| $\mathcal{ALC}, \mathcal{ALCI}, \mathcal{ALCN}$ | PSpace-Complete |

**Observations:** two sources of complexity:

- union ( $\mathcal{U}$ ) of type **NP**,
- qualified existential restriction ( $\mathcal{E}$ ) of type **coNP**.

With more combinations the complexity jumps to **PSpace**.

Number restrictions ( $\mathcal{N}$ ) do not add to the complexity.

# Structural properties vs. asserted properties

So far, we have seen how to build **complex concept and roles expressions**, which allow one to denote classes with a complex structure.

However, in order to represent real world domains, one needs the ability to **assert properties** of classes and relationships between them (e.g., as done in ER and UML class diagrams).

The assertion of properties is done in DLs by means of an **ontology** (or knowledge base).

# Description Logics ontology (or knowledge base)

A DL ontology is a pair  $\mathcal{O} = \langle \mathbf{T}, \mathbf{A} \rangle$ , where  $\mathbf{T}$  is a **TBox** and  $\mathbf{A}$  is an **ABox**

**Definition:** A DL **TBox** consists of a set of assertions on concepts and roles:

- Inclusion assertions on concepts:  $C_1 \sqsubseteq C_2$
- Inclusion assertions on roles:  $R_1 \sqsubseteq R_2$
- Property assertions on (atomic) roles such as:  
**(transitive P), (symmetric P), (functional P), (reflexive P) ...**

**Definition:** A DL **ABox** consists of a set of assertions on **individuals** (we use  $c_i$  to denote individuals):

- Membership assertions for concepts:  $D(c)$
- Membership assertions for roles:  $R(c_1, c_2)$
- Equality and distinctness assertions:  $c_1 \approx c_2; \quad c_1 \not\approx c_2$

# Description Logics ontology – Example

**TBox** assertions:

Concept inclusion assertions:

**HappyFather**  $\sqsubseteq \forall \text{descendant}.\text{(Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson})$

**HappyAnc**  $\sqsubseteq \forall \text{descendant}.\text{HappyFather}$

**Teacher**  $\sqsubseteq \neg \text{Doctor} \sqcap \neg \text{Lawyer}$

Role inclusion assertions:

**hasChild**  $\sqsubseteq \text{descendant}$

**hasFather**  $\sqsubseteq \text{hasChild}^{-}$

Property assertions on roles:

**(transitive descendant), (reflexive descendant), (functional hasFather)**

**ABox** assertions:

**Teacher(mary), hasFather(mary; john), HappyAnc(john)**

# Semantics of a DL ontology

The **semantics** is given by specifying when an interpretation  $I$  satisfies an assertion  $\alpha$ , denoted by  $I \models \alpha$

- **TBox** assertions:

$I \models C_1 \sqsubseteq C_2$  if  $C_1^I \subseteq C_2^I$

$I \models R_1 \sqsubseteq R_2$  if  $R_1^I \subseteq R_2^I$

$I \models (\text{prop } P)$  if  $P^I$  is a relation that has property **prop**

- **ABox** assertions:

We need first to extend the interpretation function  $I$ , so that it maps each individual  $c$  to an element  $c^I$  of  $\Delta^I$ .

$I \models A(c)$  if  $c^I \in A^I$

$I \models P(c_1, c_2)$  if  $(c_1^I, c_2^I) \in P^I$

$I \models c_1 \approx c_2$  if  $c_1^I = c_2^I$

$I \models c_1 \not\approx c_2$  if  $c_1^I \neq c_2^I$

# Model of a Description Logics ontology

**Definition:** An interpretation  $I$  is a **model** of:

- An assertion  $\alpha$ , if it satisfies  $\alpha$ , i.e.,  $I \models \alpha$
- A **TBox**  $T$ , if it satisfies all the assertions in  $T$
- An **ABox**  $A$ , if it satisfies all the assertions in  $A$
- An ontology  $O = \langle T, A \rangle$ , if it is a model for both  $T$  and  $A$ .

In what follows, we will use  $I \models \beta$  to denote the fact that  $I$  is a model for  $\beta$  (where  $\beta$  stands for an assertion, **TBox**, **ABox**, or **ontology**). Moreover, we will use **Mod(O)** to denote the set of models of the ontology  $O$ .

# **Reasoning in Description Logics**

## Logical implication

The fundamental reasoning service from which all other ones can be easily derived is **logical implication**.

**Definition (Logical implication):** An ontology  $\mathbf{O}$  logically implies an assertion  $\alpha$ , written  $\mathbf{O} \models \alpha$ , if  $\alpha$  is satisfied by all models of  $\mathbf{O}$ , i.e.,  $I \models \alpha, \forall I \in Mod(\mathbf{O})$ .

We can provide an analogous definition for a **TBox  $\mathbf{T}$**  instead of an ontology  $\mathbf{O}$ .

# TBox reasoning

**TBox Satisfiability:**  $T$  is satisfiable, if it admits at least one model.

**Concept Satisfiability:**  $C$  is satisfiable w.r.t.  $T$ , if there is a model  $I$  of  $T$  such that  $C^I$  is not empty, i.e.,  $T \models C \equiv \perp$

**Subsumption:**  $C_1$  is subsumed by  $C_2$  w.r.t.  $T$ , if for every model  $I$  of  $T$  we have  $C_1^I \subseteq C_2^I$ , i.e.,  $T \models C_1 \sqsubseteq C_2$

**Equivalence:**  $C_1$  and  $C_2$  are equivalent w.r.t.  $T$  if for every model  $I$  of  $T$  we have  $C_1^I = C_2^I$ , i.e.,  $T \models C_1 \equiv C_2$

**Disjointness:**  $C_1$  and  $C_2$  are disjoint w.r.t.  $T$  if for every model  $I$  of  $T$  we have  $C_1^I \cap C_2^I = \emptyset$ , i.e.,  $T \models C_1 \sqcap C_2 \equiv \perp$

**Functionality implication:** A functionality assertion (*funct R*) is logically implied by  $T$  if for every model  $I$  of  $T$ , we have that  $(o, o_1) \in R^I$  and  $(o, o_2) \in R^I$  implies  $o_1 = o_2$ , i.e.,  $T \models (\text{funct } R)$ .

Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.

# Reasoning over an ontology

**Ontology Satisfiability:** Verify whether an ontology  $O$  is satisfiable, i.e., whether  $O$  admits at least one model.

**Concept Instance Checking:** Verify whether an individual  $c$  is an instance of a concept  $C$  in every model of  $O$ , i.e., whether  $O \models C(c)$ .

**Role Instance Checking:** Verify whether a pair  $(c_1, c_2)$  of individuals is an instance of a role  $R$  in every model of  $O$ , i.e., whether  $O \models R(c_1, c_2)$ .

**Query Answering:** see later . . .

# Reasoning in Description Logics – Exercise

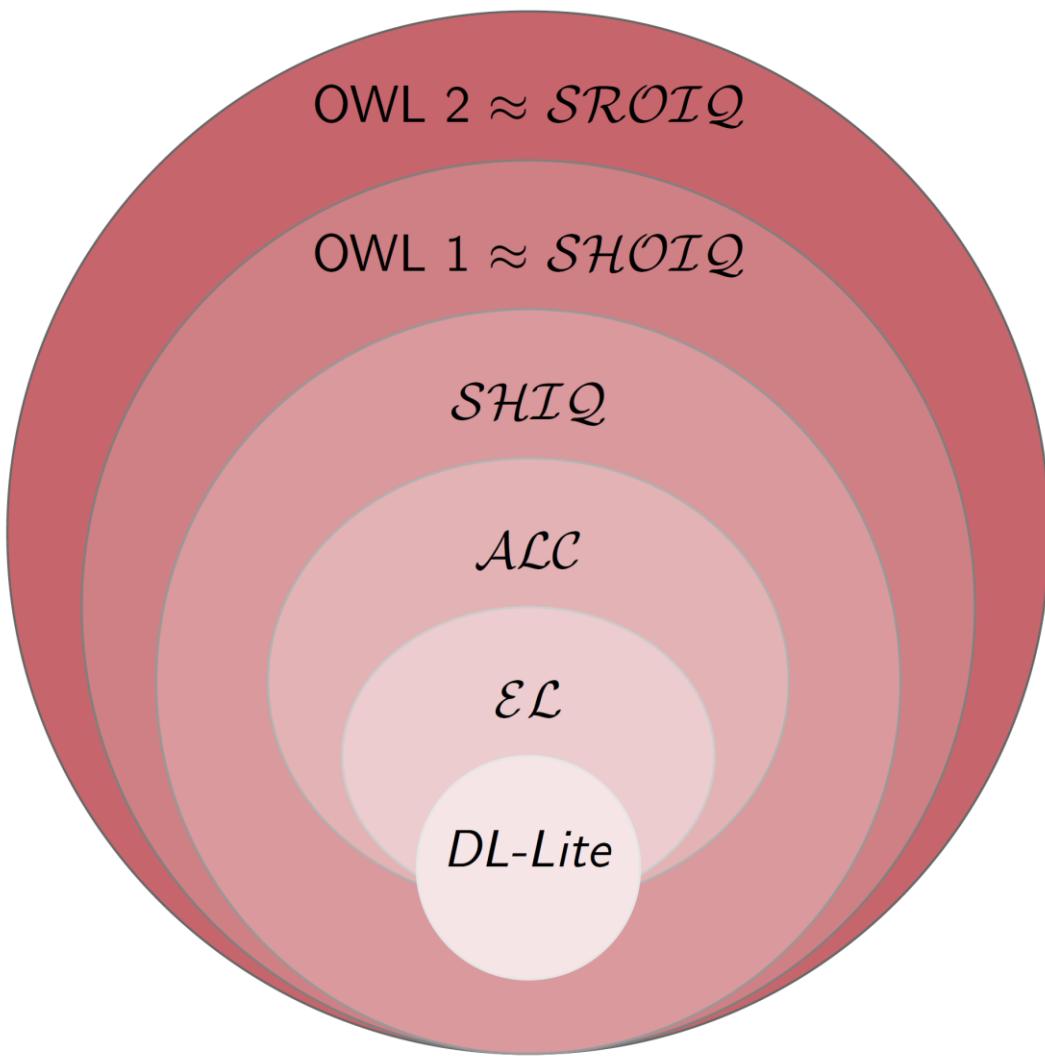
Consider the ontology  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$ , where:

TBox  $\mathbf{T} = \{ \text{HappyFather} \sqsubseteq \forall \text{descendant}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson}),$   
 $\text{HappyAnc} \sqsubseteq \forall \text{descendant}.\text{HappyFather},$   
 $\text{Teacher} \sqsubseteq \neg \text{Doctor} \sqcap \neg \text{Lawyer},$   
 $\text{hasChild} \sqsubseteq \text{descendant},$   
 $\text{hasFather} \sqsubseteq \text{hasChild}^{-},$   
(transitive descendant),  
(reflexive descendant),  
(functional hasFather) }

ABox  $\mathbf{A} = \{ \text{Teacher(mary)}, \text{hasFather(mary; john)}, \text{HappyAnc(john)} \}$

We have that:  $\mathbf{O} \vDash \text{HappyPerson(mary)}$ . Why?

# Complexity of reasoning over DL ontologies



Reasoning over DL ontologies is a **complex task**:

## Bad news:

- without restrictions on the form of TBox assertions, reasoning over DL ontologies is already **ExpTime-hard**, even for very simple DLs

## Good news:

- There are DL reasoners that perform reasonably well in practice for such DLs (e.g, Racer, Pellet, Fact++, . . . )

# DLs vs other Conceptual Modeling Languages

Most DLs are **well-behaved** fragments of **First Order Logic (FOL)** – differently from FOL theories, reasoning over a DL ontology is **decidable**.

It is possible to encode DL ontologies in FOL theories and reduce DL reasoning services to FOL reasoning services.

DLs are nowadays considered the foundations for ontology languages. **OWL (Web Ontology Language)**, the W3C standard for specifying ontology in the *Semantic Web*, has been defined as syntactic variants of certain DLs.

DLs are also ideally suited to capture the fundamental features of conceptual modeling formalism used in information systems design such as:

- **Entity-Relationship diagrams**, used in database conceptual modeling
- **UML Class Diagrams**, used in the design phase of software applications

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Query answering in ontologies

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

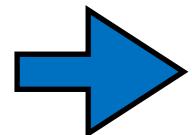
# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
4. Information Integration Systems (IIS)
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. Ontology-Based Data Management (OBDM)
10. Description Logic Ontologies
11. **Query answering in ontologies**
12. Query answering in OBDM

# Query answering in ontologies

# Query answering in tradition databases

- Data are completely specified (**Close Word Assumption**), and are typically large.
- Intensional information (the schema) is used **only** in the design phase. At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

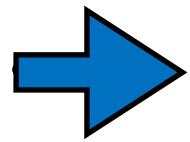


Query answering amounts to **query evaluation**, which is computationally easy.

# Query answering in ontologies

An ontology imposes constraints on the data and actual data may be **incomplete** w.r.t. such constraints (**incomplete information assumption** on the data, as in **information integration** in databases).

The system has to **take into account** the constraints during query answering, and **overcome incompleteness**.

 Query answering amounts to **logical inference**, which is computationally more costly than (classic) query answering over a database.

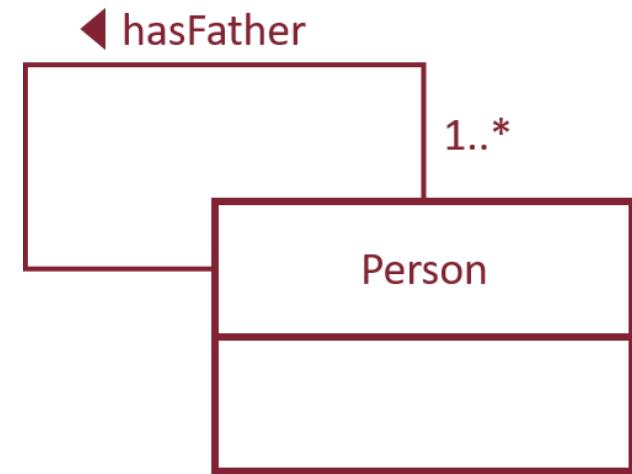
**Note:** the size of the data is not considered critical (comparable to the size of the intensional information) and queries are typically simple, i.e., atomic (a class name), and query answering amounts to instance checking.

## Query answering in ontologies – Example 1 (1\2)

Consider an ontology modelling the paternity relationships between persons:  
"each Person has a father who is a Person"

The following TBox captures our domain:

**TBox T** = { **Person**  $\sqsubseteq \exists \text{hasFather}$ ,  
 $\exists \text{hasFather} \sqsubseteq \text{Person}$   
 $\exists \text{hasFather}^- \sqsubseteq \text{Person}$  }



Suppose our ABox contains the following facts:

**ABox A** = { **Person(john)**, **Person(bob)**, **Person(tom)**,  
**hasFather(john,bob)**, **hasFather(bob,tom)** }

# Query answering in ontologies – Example 1 (2\2)

$T = \{ \text{Person} \sqsubseteq \exists \text{hasFather},$   
 $\exists \text{hasFather} \sqsubseteq \text{Person},$   
 $\exists \text{hasFather}^- \sqsubseteq \text{Person} \}$

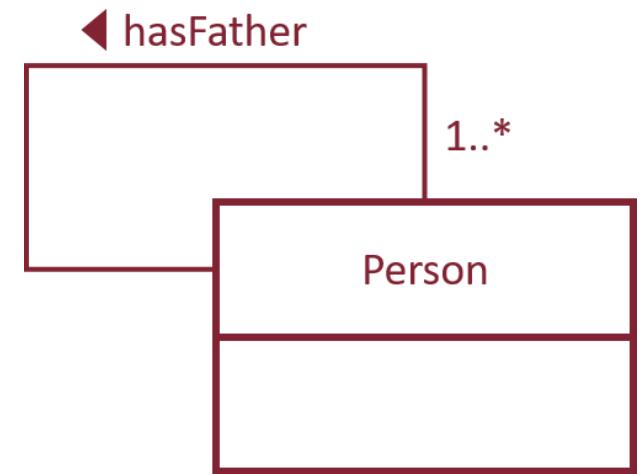
$A = \{ \text{Person(john)}, \text{Person(bob)}, \text{Person(tom)},$   
 $\text{hasFather(john,bob)}, \text{hasFather(bob,tom)} \}$

Queries:

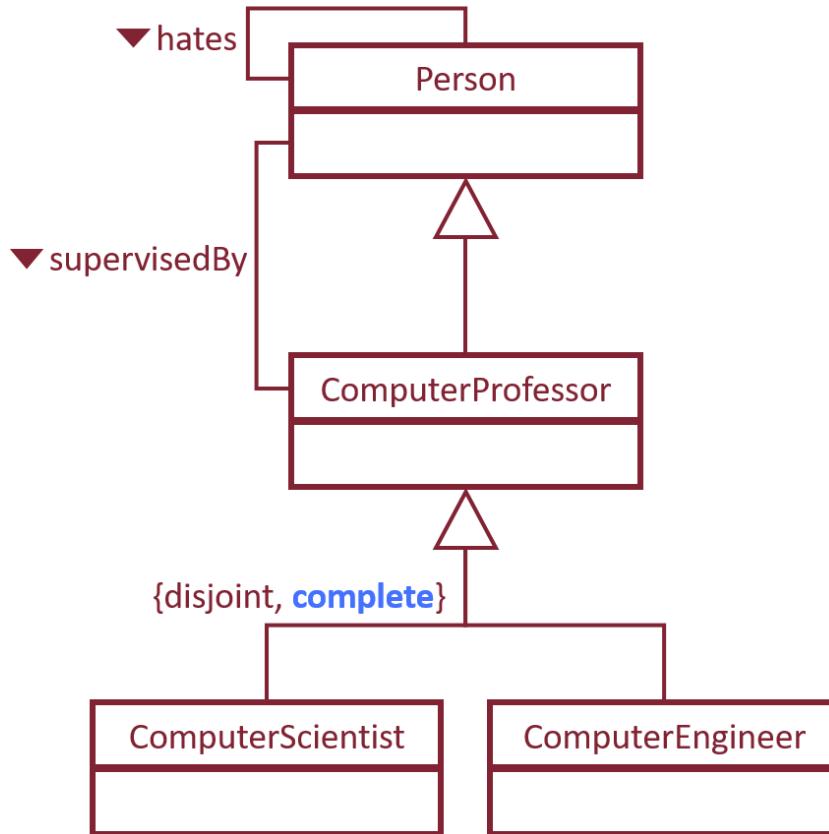
- $q_1: \{(x, y) \mid \text{hasFather}(x, y)\}$
- $q_2: \{(x) \mid \exists y. \text{hasFather}(x, y)\}$
- $q_3: \{(x) \mid \exists y, v, w. \text{hasFather}(x, y) \wedge \text{hasFather}(y, v) \wedge \text{hasFather}(v, w)\}$
- $q_4: \{(x, w) \mid \exists y, v. \text{hasFather}(x, y) \wedge \text{hasFather}(y, v) \wedge \text{hasFather}(v, w)\}$

Answers:

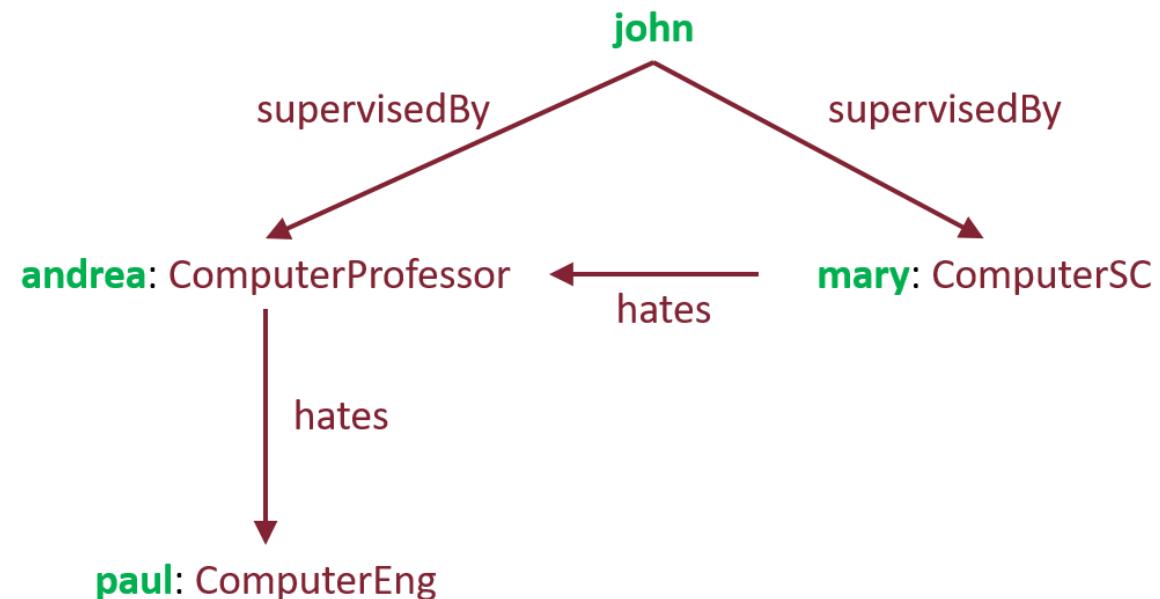
- to  $q_1: \{ (\text{john}, \text{bob}), (\text{bob}, \text{tom}) \}$
- to  $q_2: \{ \text{john}, \text{bob}, \text{tom} \}$
- to  $q_3: \{ \text{john}, \text{bob}, \text{tom} \}$
- to  $q_4: \{ \}$



## Query answering in ontologies – Example 2 (2\2)



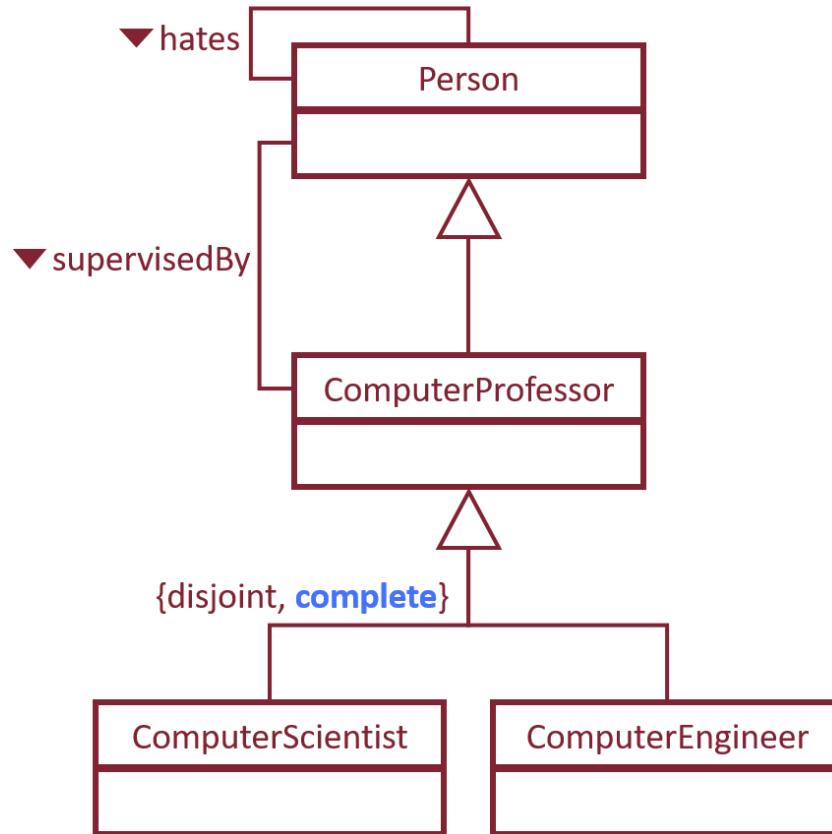
Note that **ComputerProfessor** is partitioned into **ComputerScientist** and **ComputerEngineer**.



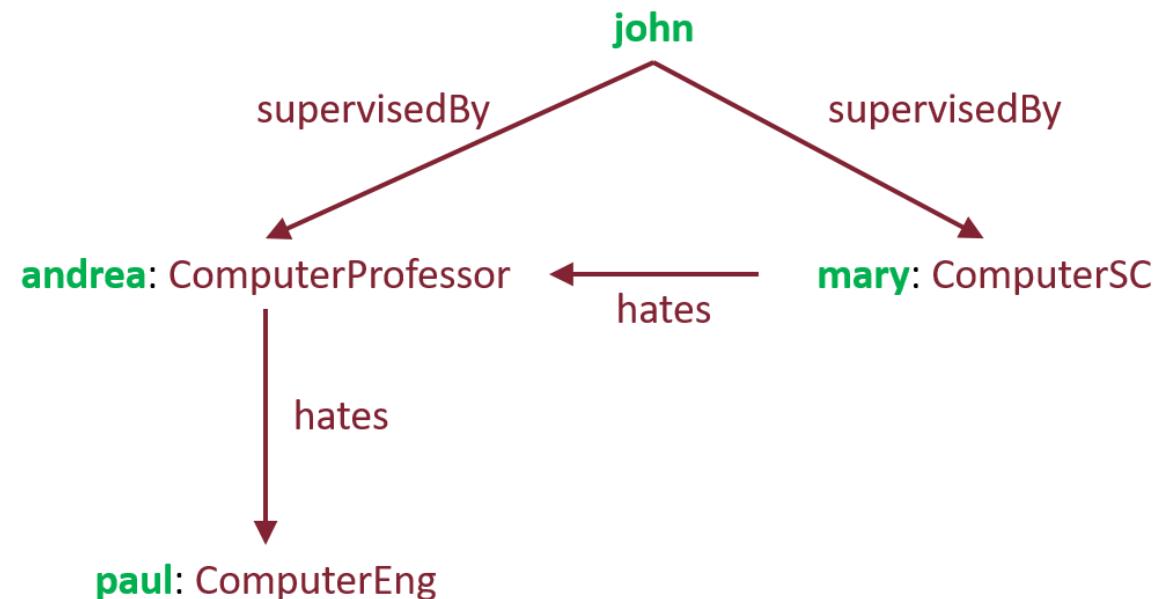
**Query:**  $\{ (x) \mid \exists y, z. \text{supervisedBy}(x, y) \wedge \text{ComputerSC}(y) \wedge \text{hates}(y, z) \wedge \text{ComputerEng}(z) \}$

**Answer:** ???

## Query answering in ontologies – Example 2 (2\2)



Note that **ComputerProfessor** is partitioned into **ComputerScientist** and **ComputerEngineer**.



**Query:**  $\{ (x) \mid \exists y, z. \text{supervisedBy}(x, y) \wedge \text{ComputerSC}(y) \wedge \text{hates}(y, z) \wedge \text{ComputerEng}(z) \}$

**Answer:** { **john** } → To get this answer we need to resort to reasoning by cases.

# Query answering in ontologies

Two borderline cases:

- Just **classes and properties** of the ontology → **instance checking**
  - Ontology languages are tailored for capturing intensional relationships.
  - They are **quite poor as query languages**: we cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of join, namely chaining.
- Full **SQL** (or equivalently, **first-order logic**)
  - Problem: in the presence of **incomplete information**, query answering becomes **undecidable** (FOL validity).
- **A good tradeoff is to use (unions of) conjunctive queries.**

## Conjunctive Query (recap)

Let  $\varphi(x_1, \dots, x_n)$  be a FOL formula with free variables  $x_1, \dots, x_n$

$\varphi(x_1, \dots, x_n)$  is an **Existential Conjunction** if it is of the form

$$\exists y_1, \dots, \exists y_n P_1(\bar{z}_1) \wedge \dots \wedge P_k(\bar{z}_k)$$

A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Conjunctive Query (CQ)** if  $\varphi(x_1, \dots, x_n)$  is an Existential Conjunction.

**NOTE:** In a **CQ** over a DL ontology the predicates  $P_i$  in the Existential Conjunction are only **atomic concepts** (unary predicates) and **atomic roles** (binary predicates)

## Union of Conjunctive Queries (recap)

- A formula  $\varphi(x_1, \dots, x_n)$  is Union of Existential Conjunctions if it is of the form

$$\bigvee_i \psi_i(x_1, \dots, x_n)$$

where each  $\psi_i$  is an existential conjunction.

**Observe:** same free variables.

- A FOL query  $\{(x_1, \dots, x_n) | \varphi(x_1, \dots, x_n)\}$  is a **Union of Conjunctive Queries** if  $\varphi(x_1, \dots, x_n)$  is a Union of Existential Conjunctions.

## Certain answers to a query

Let  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  be an **ontology**,  $I$  an **interpretation** for  $\mathbf{O}$ , and  $\mathbf{q} = \{\bar{x} \mid \exists \bar{y}. \varphi(\bar{x}, \bar{y})\}$  be a **CQ**.

- **Definition:** The answer to  $\mathbf{q}$  over  $I$ , denoted  $q^I$ , is the set of **tuples  $\bar{c}$  of constants of  $\mathbf{A}$**  such that the formula  $\exists \bar{y}. \varphi(\bar{c}, \bar{y})$  evaluates to true in  $I$ .

We are interested in finding those answers that hold in **all models of an ontology**.

- **Definition:** The **certain answers** to  $\mathbf{q}$  over  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$ , denoted  $\text{cert}(\mathbf{q}, \mathbf{O})$  are the tuples  $\bar{c}$  of constants of  $\mathbf{A}$  such that  $\bar{c} \in q^I$ , **for every model  $I$  of  $\mathbf{O}$ , i.e.,  $\forall I \in \text{Mod}(\mathbf{O})$**

# Query answering in ontologies

- **Definition:** **Query answering** over an ontology  $O$  is the problem of computing the **certain answer** to a query over  $O$

Computing certain answers is a form of **logical implication**:

$$\bar{c} \in \text{cert}(\{\bar{x} \mid \exists \bar{y}. \varphi(\bar{x}, \bar{y})\}, O) \quad \text{iff} \quad O \models \exists \bar{y}. \varphi(\bar{c}, \bar{y})$$

**Note:** A special case of query answering is **instance checking**: it amounts to answering the Boolean query  $q = \{() \mid A(c)\}$  (resp.,  $q = \{() \mid P(c, c')\}$ ) over  $O$ .

**Remember:** in case of Boolean queries the answer is the **empty set** (negative answer) or the **empty tuple** (positive answer).

# Query answering in ontologies – Example 1 (2\2)

$T = \{ \text{Person} \sqsubseteq \exists \text{hasFather},$   
 $\exists \text{hasFather} \sqsubseteq \text{Person},$   
 $\exists \text{hasFather}^- \sqsubseteq \text{Person} \}$

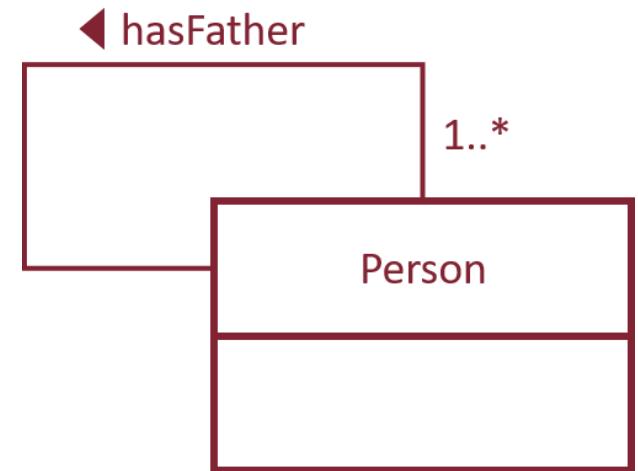
$A = \{ \text{Person(john)}, \text{Person(bob)}, \text{Person(tom)},$   
 $\text{hasFather(john,bob)}, \text{hasFather(bob,tom)} \}$

Queries:

- $q_1: \{(x, y) \mid \text{hasFather}(x, y)\}$
- $q_2: \{(x) \mid \exists y. \text{hasFather}(x, y)\}$
- $q_3: \{(x) \mid \exists y, v, w. \text{hasFather}(x, y) \wedge \text{hasFather}(y, v) \wedge \text{hasFather}(v, w)\}$
- $q_4: \{(x, w) \mid \exists y, v. \text{hasFather}(x, y) \wedge \text{hasFather}(y, v) \wedge \text{hasFather}(v, w)\}$

Certain answers:

- $\text{cert}(q_1, O): \{ (\text{john}, \text{bob}), (\text{bob}, \text{tom}) \}$
- $\text{cert}(q_2, O): \{ \text{john}, \text{bob}, \text{tom} \}$
- $\text{cert}(q_3, O): \{ \text{john}, \text{bob}, \text{tom} \}$
- $\text{cert}(q_4, O): \{ \}$



# Complexity measures for queries over ontologies

When measuring the complexity of answering a query  $q$  over an ontology  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$ , various parameters are of importance.

Depending on which parameters we consider, we get different complexity measures:

- **Data complexity**: only the size of the **ABox** (i.e., the data) matters. **TBox** and **query** are considered fixed.
- **Query complexity**: only the size of the query matters. **TBox** and **ABox** are considered fixed.
- **Schema complexity**: only the size of the **TBox** (i.e., the schema) matters. **ABox** and **query** are considered fixed.
- **Combined complexity**: no parameter is considered fixed.

**Note:** In the **OBDM setting**, the size of the **data** largely dominates the size of the conceptual layer (and of the query) → **Data complexity is the relevant complexity measure.**

# Recognition problem for query answering

When studying the complexity of **query answering**, we need to consider the associated **decision problem**:

**Definition [Recognition problem for query answering over an ontology]:**

Given an ontology  $O = \langle T, A \rangle$ , a query  $q$  over  $O$ , and a tuple  $\bar{c}$  of constants, check whether  $\bar{c} \in \text{cert}(q, O)$ .

We look mainly at the data complexity of query answering, i.e., complexity of the recognition problem computed w.r.t. the size of the **ABox  $A$**  only.

# Complexity of query answering in DLs

Studied extensively for (unions of) CQs and various ontology languages:

|                                                       | Combined complexity | Data complexity     |
|-------------------------------------------------------|---------------------|---------------------|
| Plain databases                                       | NP-complete         | AC <sup>0</sup> (1) |
| $\mathcal{ALCI}, \mathcal{SH}, \mathcal{SHIQ}, \dots$ | 2ExpTime-complete   | coNP-complete (2)   |
| OWL 2                                                 | 3ExpTime-hard       | coNP-hard           |

(1) This is what we need to scale with the data.

(2) coNP-hard already for a TBox with a single disjunction

## Questions:

- Can we find interesting description logics for which query answering can be done efficiently (i.e., in AC<sup>0</sup>)?
- If yes, can we leverage **relational database technology** for query answering? (We need this for OBDM)

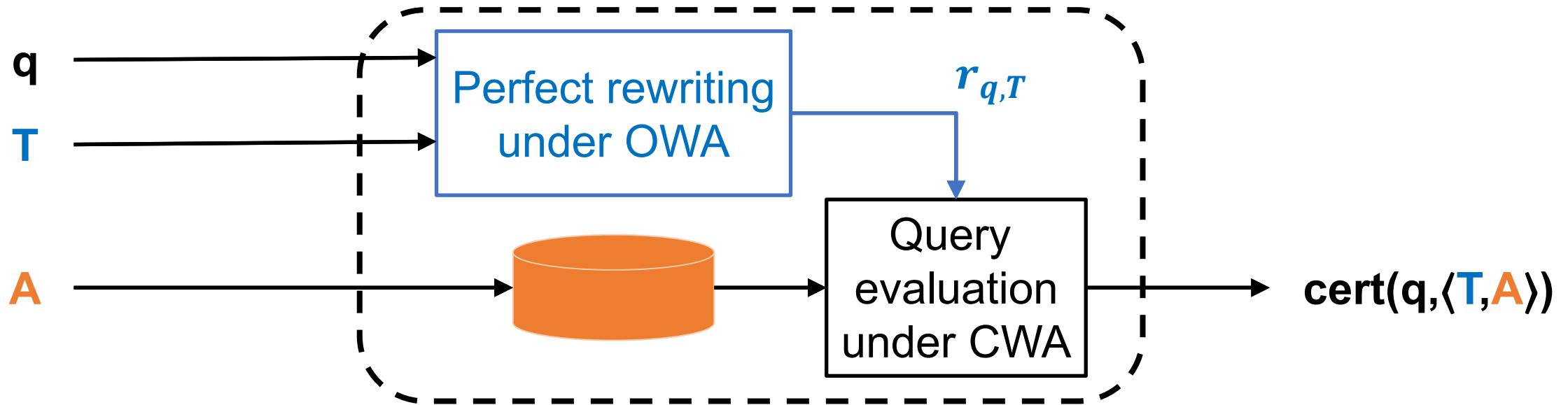
## Inference in query answering



To be able to deal with data efficiently, we need to separate the contribution of  $A$  from the contribution of  $q$  and  $T$

→ Query answering by **query rewriting**.

# Query answering by query rewriting



Query answering can be thought as done in two phases:

- **Perfect rewriting**: produce from  $q$  and the TBox  $T$  a new query  $r_{q,T}$  (called the **perfect rewriting** of  $q$  w.r.t.  $T$  ).
- **Query evaluation**: evaluate  $r_{q,T}$  over the ABox  $A$  seen as a complete database (without considering the TBox  $T$ ). The result of such evaluation is  $\text{cert}(q, \langle T, A \rangle)$

# $\mathcal{Q}$ -rewritability

Let  $\mathcal{Q}$  be a query language and  $\mathcal{L}$  an ontology language.

- **Definition [ $\mathcal{Q}$ -rewritability]:** For an ontology language  $\mathcal{L}$ , query answering is  $\mathcal{Q}$ -**rewritable** if for every TBox  $\mathbf{T}$  of  $\mathcal{L}$  and for every query  $\mathbf{q}$ , the perfect reformulation  $r_{q,T}$  of  $\mathbf{q}$  w.r.t.  $\mathbf{T}$  can be expressed in the query language  $\mathcal{Q}$ .

Notice that the complexity of computing  $r_{q,T}$  or the size of  $r_{q,T}$  do **not** affect data complexity.

Hence,  $\mathcal{Q}$ -**rewritability** is tightly related to **data complexity**, i.e.:

- the complexity of computing  $\text{cert}(\mathbf{q}, \langle \mathbf{T}, \mathbf{A} \rangle)$  measured in the size of the ABox  $\mathbf{A}$  only,
- which corresponds to the **complexity of evaluating  $r_{q,T}$  over  $\mathbf{A}$  (seeing  $\mathbf{A}$  as a database)**

## *Q*-rewritability and OBDM

The expressiveness of the ontology language affects the rewriting language, i.e., the language into which we are able to rewrite UCQs:

We are especially interested in **FOL-rewritability**:

- The rewriting  $r_{q,T}$  can be expressed in FOL, i.e., in **SQL**.
- Recall evaluating FOL\SQL queries over a **plain database** is in **AC<sup>0</sup>** in data complexity.
- Query evaluation can be **delegated to a relational DBMS**.

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.

# Advanced Data Management

## Query answering in OBDM

Domenico Fabio Savo

*Corso di laurea magistrale  
INGEGNERIA INFORMATICA*

*Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione  
University of Bergamo*

# Outline of the course

1. Introduction to propositional logic
2. Introduction to First-order Logic
3. Relational Calculus
- 4. Information Integration Systems (IIS)**
5. Logical formalization of IISs
6. Mapping between **Global Schema** e **Data Sources**
7. Incomplete information databases
8. Query answering over IISs
9. Ontology-Based Data Management (OBDM)
10. Description Logic Ontologies
11. Query answering in ontologies
- 12. Query answering in OBDM**

# **Ontology-based Data Management**

# Query answering in ontology-based data management

In OBDM, we have to face the following difficulties:

- The actual data is stored in external information sources (i.e., databases), and thus its size is typically **very large**.
- The ontology introduces **incompleteness** of information, and we have to do **logical inference**, rather than query evaluation.
- We want to take into account at **runtime** the **constraints** expressed in the ontology.
- We want to answer **complex database-like queries**.
- We may have to deal with **multiple information sources**, and thus face also the problems that are typical of data integration.

# Questions that need to be addressed

In the context of ontology-based data management:

- Which is the "right" (user's) **query language**?
- Which is the "right" **ontology language**?
- How can we bridge the **semantic mismatch** between the ontology and the data sources? (**in an ontology we have objects of the domain while in a database we have values**)

# OBDM: user's query language

As already shown, we have two borderline cases:

- Just **classes and properties** of the ontology → **instance checking**
  - They are **quite poor as query languages**: we cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of join, namely chaining.
- Full **SQL** (or equivalently, **first-order logic**)
  - Problem: in the presence of **incomplete information**, query answering becomes **undecidable** (FOL validity).
- **A good tradeoff is to use (unions of) conjunctive queries.**

**So, we fix (U)CQ as user's query language, i.e., the query we pose over the OBDM system.**

# Questions that need to be addressed

In the context of ontology-based data management:

- Which is the "right" (user's) **query language** → UCQ
- Which is the "right" **ontology language**?
- How can we bridge the **semantic mismatch** between the ontology and the data sources? (**in an ontology we have objects of the domain while in a database we have values**)

# The DL-Lite Family

DL-Lite is a family of DLs optimized according to the **tradeoff** between **expressive power** and **complexity** of query answering, with emphasis on **data**.

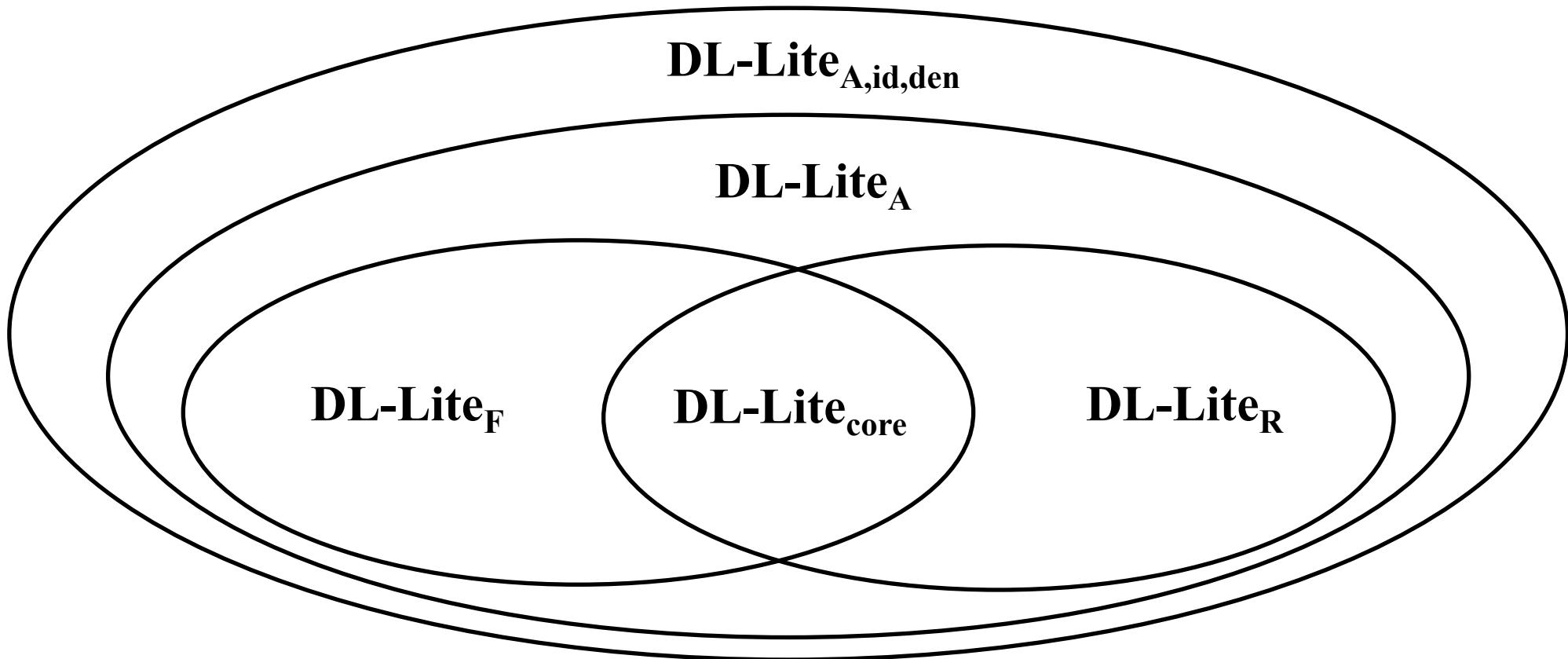
Carefully designed to have **nice computational properties** for answering UCQs (i.e., computing certain answers):

- The same data complexity as **relational databases**.
- In fact, query answering can be delegated to a **relational DB engine**.
- The DLs of the DL-Lite family are essentially the **maximally expressive ontology languages** enjoying these nice computational properties.
- Captures **conceptual modeling formalism**.

**DL-Lite provides robust foundations for Ontology-Based Data Management.**

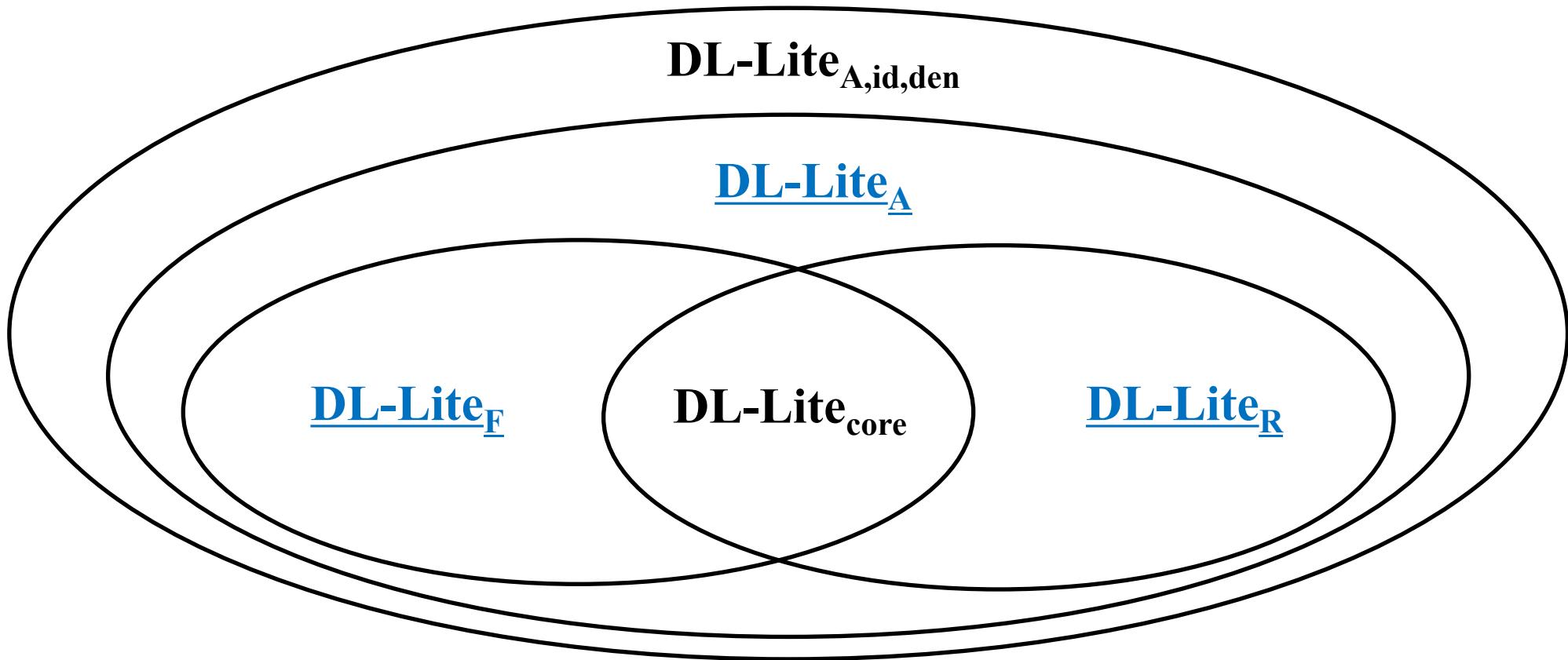
# Members of the DL-Lite Family

The main members of the **DL-Lite** is a family are:



# Members of the DL-Lite Family

We focus on the following:



# Basic features of DL-Lite<sub>A</sub>

DL-Lite<sub>A</sub> is an expressive member of the **DL-Lite family**

- Takes into account the distinction between **objects** and **values**:
  - Objects are elements of an abstract interpretation domain (the instances of concepts).
  - Values are elements of concrete data types, such as integers, strings, ecc. Values are connected to objects through **attributes**.
- Captures most of **UML** class diagrams and **ER** diagrams.
- Enjoys **nice computational properties**, both w.r.t. the traditional reasoning tasks, and w.r.t. query answering (see later).

# Syntax of the DL-Lite<sub>A</sub> description language

## Role expressions:

- atomic role:  $P$
- basic role:  $Q ::= P \mid P^-$  (where  $P^-$  denotes the inverse of the atomic role  $P$ )

## Concept expressions:

- atomic concept:  $A$
- basic concept:  $B ::= A \mid \exists Q \mid \delta(U)$

(where  $\exists Q$  denotes the *domain* of the basic role  $Q$ , i.e., set of objects participating to  $Q$  as first element, while  $\delta(U)$  denotes the *domain* of the attribute  $U$ , i.e., the set of objects that  $U$  relates to values)

## Attribute expressions:

- atomic attribute:  $U$

## Value-domain expressions:

- attribute range:  $\rho(U)$  (i.e., the set of values that the attribute  $U$  relates to objects)
- (RDF) datatypes:  $T_i$  (such as xsd:string, xsd:integer, xsd:dateTime, etc.)

# Concept constructors

| Construct                | Syntax      | Example                 | Semantics                                    |
|--------------------------|-------------|-------------------------|----------------------------------------------|
| <b>Atomic concept</b>    | $A$         | Student                 | $A^I \subseteq \Delta_O^I$                   |
| <b>Atomic role</b>       | $P$         | LivesIn                 | $P^I \subseteq \Delta_O^I \times \Delta_O^I$ |
| <b>Atomic attribute</b>  | $U$         | Salary                  | $U^I \subseteq \Delta_O^I \times \Delta_V^I$ |
| <b>Datatype</b>          | $T_i$       | xsd:integer             | $T_i^I \subseteq \Delta_V^I$ (predefined)    |
| <b>Inverse role</b>      | $P^-$       | LivesIn $^-$            | $\{(o', o) \mid (o, o') \in P^I\}$           |
| <b>Exis. restriction</b> | $\exists R$ | $\exists$ LivesIn       | $\{o \mid \exists o'. (o, o') \in R^I\}$     |
| <b>Attribute domain</b>  | $\delta(U)$ | $\delta(\text{salary})$ | $\{o \mid \exists v. (o, v) \in U^I\}$       |
| <b>Attribute range</b>   | $\rho(U)$   | $\rho(\text{salary})$   | $\{v \mid \exists o. (o, v) \in U^I\}$       |

Where  $\Delta_O^I$  denotes the domain of objects, while  $\Delta_V^I$  denotes the domain of values.

# DL-Lite<sub>A</sub> assertions

TBox assertions can have the following forms:

## Inclusion assertions (a.k.a. positive inclusions)

|                   |                       |                        |                           |
|-------------------|-----------------------|------------------------|---------------------------|
| Concept inclusion | $B_1 \sqsubseteq B_2$ | Attribute inclusion    | $U_1 \sqsubseteq U_2$     |
| Role inclusion    | $Q_1 \sqsubseteq Q_2$ | Value-domain inclusion | $\rho(U) \sqsubseteq T_i$ |

## Disjointness assertions (a.k.a negative inclusions)

|                      |                            |                        |                       |
|----------------------|----------------------------|------------------------|-----------------------|
| Concept disjointness | $B_1 \sqsubseteq \neg B_2$ | Attribute disjointness | $U_1 \sqsubseteq U_2$ |
| Role disjointness    | $P_1 \sqsubseteq P_2$      |                        |                       |

## Functionality assertions

|                    |           |                         |           |
|--------------------|-----------|-------------------------|-----------|
| Role functionality | (funct Q) | Attribute functionality | (funct U) |
|--------------------|-----------|-------------------------|-----------|

---

ABox assertions (a.k.a. membership assertions) have the following forms:

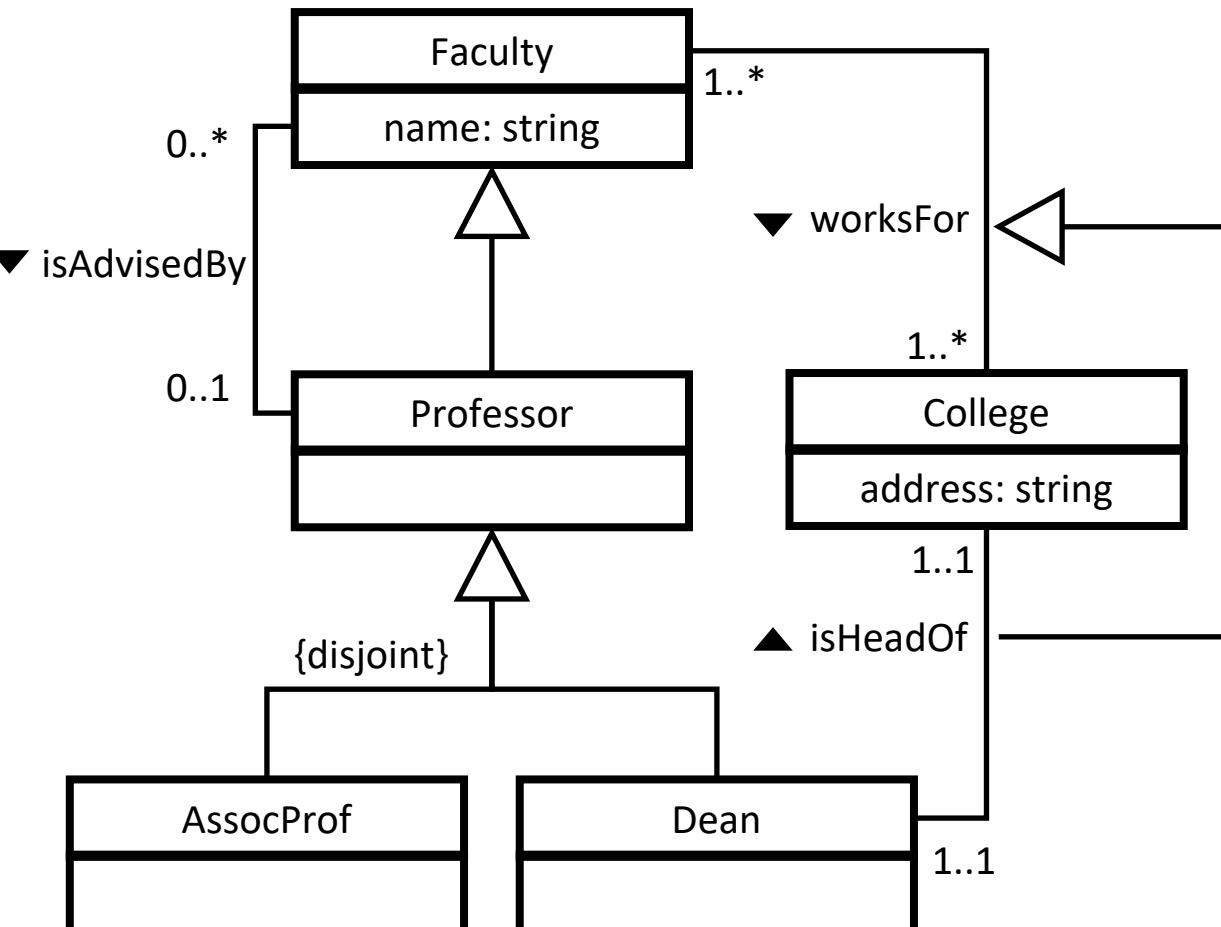
$A(c)$      $P(c,c')$      $U(c,v)$

where  $c, c'$  are object constants and  $v$  is a value constant.

# Semantics of a DL ontology

|                         | Syntax                     | Semantics                                                                             | Example                                           |
|-------------------------|----------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------|
| Concept inclusion       | $B_1 \sqsubseteq B_2$      | $B_1^I \sqsubseteq B_2^I$                                                             | Person $\sqsubseteq \exists \text{livesIn}$       |
| Role inclusion          | $Q_1 \sqsubseteq Q_2$      | $Q_1^I \sqsubseteq Q_2^I$                                                             | hasFather $\sqsubseteq \text{hasParent}$          |
| Attribute inclusion     | $U_1 \sqsubseteq U_2$      | $U_1^I \sqsubseteq U_2^I$                                                             | cellNumber $\sqsubseteq \text{contactInfo}$       |
| Value-domain inclusion  | $\rho(U) \sqsubseteq T_i$  | $(\rho(U))^I \subseteq T_i$                                                           | $\rho(\text{name}) \sqsubseteq \text{xsd:string}$ |
| Concept disjointness    | $B_1 \sqsubseteq \neg B_2$ | $B_1^I \cap B_2^I = \emptyset$                                                        | Car $\sqsubseteq \neg \text{Bike}$                |
| Role disjointness       | $P_1 \sqsubseteq P_2$      | $P_1^I \cap P_2^I = \emptyset$                                                        | hasFather $\sqsubseteq \neg \text{hasMother}$     |
| Attribute disjointness  | $U_1 \sqsubseteq U_2$      | $U_1^I \cap U_2^I = \emptyset$                                                        | offPhone $\sqsubseteq \neg \text{homePhone}$      |
| Role functionality      | $(\text{funct } Q)$        | $\forall o, o_1, o_2. (o. o_1) \in Q^I \wedge (o, o_2) \in Q^I \rightarrow o_1 = o_2$ | $(\text{funct hasFather})$                        |
| Attribute functionality | $(\text{funct } U)$        | $\forall o, v_1, v_2. (o. v_1) \in U^I \wedge (o, v_2) \in U^I \rightarrow v_1 = v_2$ | $(\text{funct SSN})$                              |
| Con. membership asser.  | $A(c)$                     | $c^I \in A^I$                                                                         | Person(bob)                                       |
| Role. membership asser. | $P(c_1, c_2)$              | $(c_1^I, c_2^I) \in P^I$                                                              | hasParent(bob,sam)                                |
| Attr. membership asser. | $U(c, v)$                  | $(c^I, \text{val}(v)) \in U^I$                                                        | name(bob, 'bob')                                  |

# Example of DL-Lite<sub>A</sub> TBox



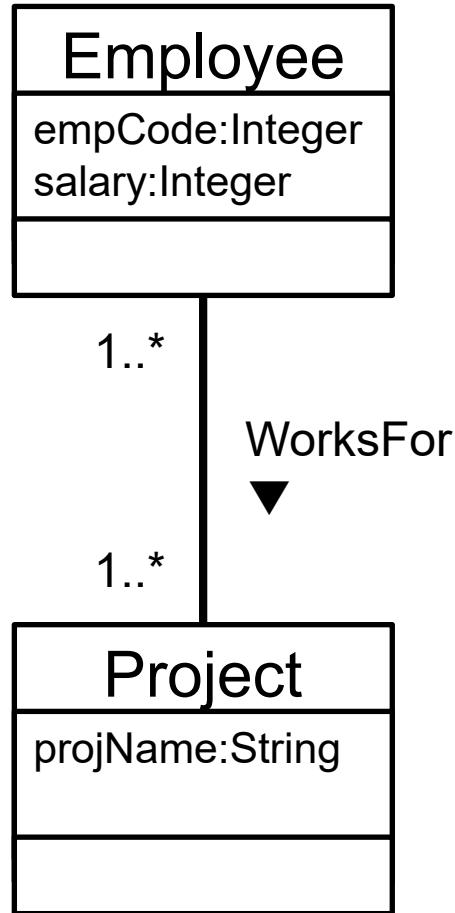
$\exists \text{worksFor}^- \sqsubseteq \text{College}$   
 $\text{isHeadOf} \sqsubseteq \text{worksFor}$   
 $\text{AssocProf} \sqsubseteq \neg \text{Dean}$   
 $\text{Professor} \sqsubseteq \text{Faculty}$   
 $\text{Dean} \sqsubseteq \exists \text{isHeadOf}$   
 $\text{College} \sqsubseteq \exists \text{isHeadOf}^-$   
 $\exists \text{isHeadOf} \sqsubseteq \text{Dean}$   
 $\exists \text{isHeadOf}^- \sqsubseteq \text{College}$   
 $\text{Faculty} \sqsubseteq \exists \text{worksFor}$   
 $\text{College} \sqsubseteq \exists \text{worksFor}^-$   
 $\exists \text{isAdvisedBy} \sqsubseteq \text{Faculty}$   
 $\exists \text{isAdvisedBy}^- \sqsubseteq \text{Professor}$  (funct isHeadOf)  
 $\exists \text{worksFor} \sqsubseteq \text{Faculty}$

$\exists \text{worksFor}^- \sqsubseteq \text{College}$   
 $\text{isHeadOf} \sqsubseteq \text{worksFor}$   
 $\text{AssocProf} \sqsubseteq \neg \text{Dean}$   
 $\text{Faculty} \sqsubseteq \delta(\text{name})$   
 $\rho(\text{name}) \sqsubseteq \text{string}$   
 $\delta(\text{name}) \sqsubseteq \text{Faculty}$   
 $\text{College} \sqsubseteq \delta(\text{address})$   
 $\rho(\text{address}) \sqsubseteq \text{string}$   
 $\delta(\text{address}) \sqsubseteq \text{College}$   
 $(\text{funct isAdvisedBy})$   
 $(\text{funct isHeadOf})$   
 $\exists \text{worksFor} \sqsubseteq \text{Faculty}$

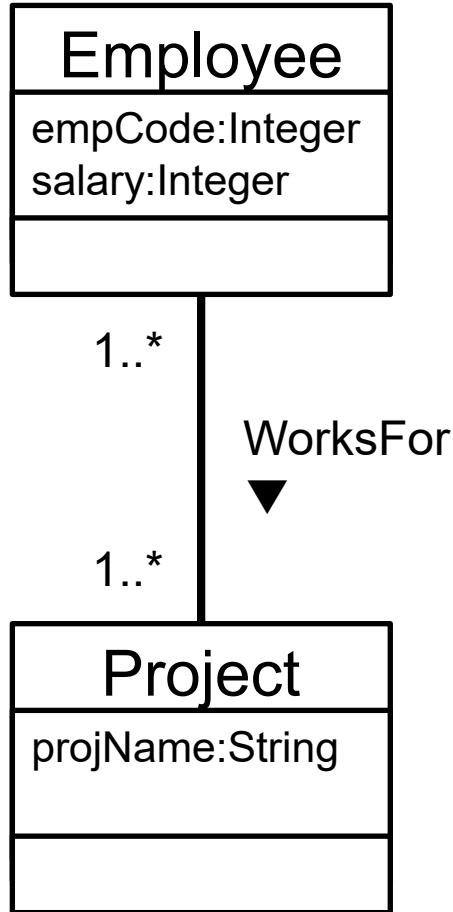
**NOTE:** DL-Lite **cannot** capture **completeness** of a hierarchy. This would require **disjunction (OR)**.

# DL-Lite<sub>A</sub> TBox: Exercise

*Model the UML diagram with a DL-Lite<sub>A</sub> TBox*



# DL-Lite<sub>A</sub> TBox: Exercise – Solution



*Model the UML diagram with a DL-Lite<sub>A</sub> TBox*

$Employee \sqsubseteq \exists WorksFor$   
 $Project \sqsubseteq \exists WorksFor^-$   
 $\exists WorksFor \sqsubseteq Employee$   
 $\exists WorksFor^- \sqsubseteq Project$   
 $Employee \sqsubseteq \delta(empCode)$   
 $Employee \sqsubseteq \delta(salary)$   
 $Project \sqsubseteq \delta(projName)$   
 $\delta(empCode) \sqsubseteq Employee$   
 $\delta(empCode) \sqsubseteq Employee$   
 $\delta(projName) \sqsubseteq Project$   
 $\rho(empCode) \sqsubseteq xsd:integer$   
 $\rho(salary) \sqsubseteq xsd:integer$   
 $\rho(projName) \sqsubseteq xsd:string$   
 $(\text{funct } empCode)$   
 $(\text{funct } salary)$   
 $(\text{funct } projName)$

# Restrictions and assumptions on DL-Lite

To ensure the good computational properties that we aim at, we have to:

Impose a **restriction** on the use of **functionality** and role/attribute inclusions.

- **Restriction on DL-Lite<sub>A</sub> TBoxes:** No functional role or attribute can be specialized by using it in the right-hand side of a role or attribute inclusion assertion, that is if (funct  $P$ ) or (funct  $P^-$ ) is in  $\mathbf{T}$ , then  $Q \sqsubseteq P$  and  $Q \sqsubseteq P^-$  are cannot be  $\mathbf{T}$  and If (funct  $U$ ) is in  $\mathbf{T}$ , then  $U' \sqsubseteq U$  cannot be  $\mathbf{T}$

Make some **assumptions** on how **individuals** are interpreted.

- **Unique name assumption (UNA):** When  $c_1$  and  $c_2$  are two individuals such that  $c_1 \neq c_2$  then  $c_1^I \neq c_2^I$ .

**Note that** when the **UNA** holds (as in DL-Lite), **functionality** cannot be used to derive new information, they can be considered as **constraints**

## **DL-Lite<sub>F</sub> and DL-Lite<sub>R</sub> of the DL-Lite Family**

We consider also two sub-languages of **DL-Lite<sub>A</sub>** (that trivially obey the previous restriction):

- **DL-Lite<sub>F</sub>**: Allows for functionality assertions, but does not allow for role inclusion assertions.
- **DL-Lite<sub>R</sub>**: Allows for role inclusion assertions, but does not allow for functionality assertions.

In both **DL-Lite<sub>F</sub>** and **DL-Lite<sub>R</sub>** we **do not** consider **data values** (and hence drop value domains and attributes).

**Note:** in what follows, we will simply use **DL-Lite** to refer to any of the logics of the **DL-Lite family**.

# Capturing basic ontology constructs in DL-Lite<sub>A</sub>

|                                      |                                               |                             |
|--------------------------------------|-----------------------------------------------|-----------------------------|
| ISA between classes                  | $A_1 \sqsubseteq A_2$                         |                             |
| Disjointness between classes         | $A_1 \sqsubseteq \neg A_2$                    |                             |
| Mandatory participation to relations | $A \sqsubseteq \exists P$                     | $A \sqsubseteq \exists P^-$ |
| Domain and range of relations        | $\exists P \sqsubseteq A$                     | $\exists P^- \sqsubseteq A$ |
| Functionality of relations           | $(\text{funct } P) \quad (\text{funct } P^-)$ |                             |
| ISA between relations                | $Q_1 \sqsubseteq Q_2$                         |                             |
| Disjointness between relations       | $Q_1 \sqsubseteq \neg Q_2$                    |                             |
| Domain and range of attributes       | $\delta(U) \sqsubseteq A$                     | $\rho(U) \sqsubseteq T_i$   |
| Mandatory and functional attributes  | $A \sqsubseteq \delta(U)$                     | $(\text{funct } U)$         |

## Properties of DL-Lite

The TBox may contain **cyclic dependencies** (which typically increase the computational complexity of reasoning).

**Example:**  $A \sqsubseteq \exists P$       $\exists P \sqsubseteq A$

In the syntax, we have not included  $\sqcap$  on the right hand-side of inclusion assertions, but it can trivially be added, since  $B \sqsubseteq C_1 \sqcap C_2$  is equivalent to  $B \sqsubseteq C_1; B \sqsubseteq C_2$

A **domain** assertion on role P has the form:  $\exists P \sqsubseteq A$

A **range** assertion on role P has the form:  $\exists P^- \sqsubseteq A$

# Observations on DL-Lite<sub>A</sub>

- Captures all the basic constructs of **UML Class Diagrams** and of the **ER Model** . . .  
    . . . **except covering constraints** in generalizations.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Is at the basis of the **OWL 2 QL** profile of **OWL 2**.

# Complexity of reasoning in DL-Lite<sub>A</sub>

As shown, DL-Lite<sub>A</sub> captures the main features of main **conceptual modeling formalisms**.

We will show that DL-Lite is characterized by the following computational properties:

- **Ontology satisfiability** and all classical DL reasoning tasks are:
  - Efficiently tractable in the size of the **TBox** (i.e., **PTime**).
  - Very efficiently tractable in the size of the **ABox** (i.e., **AC<sup>0</sup>**).
- **Query answering** for CQs and UCQs is:
  - **PTime** in the size of the **TBox**.
  - **AC<sup>0</sup>** in the size of the **ABox**.
  - Exponential in the size of the **query (NP-complete)**.
  - **Bad? . . . not really, this is exactly as in relational DBs.**

DL-Lite is essentially the maximal DL enjoying these nice computational properties.

From the observations above we get that: DL-Lite is a DL very well suited to underlie **ontology-based data management systems!**

# Questions that need to be addressed

In the context of ontology-based data management:

- Which is the "right" (user's) **query language** → UCQ
- Which is the "right" **ontology language** → DL-Lite
- How can we bridge the **semantic mismatch** between the ontology and the data sources? (in an ontology we have objects of the domain while in a database we have values)

# **Reasoning and query answering in DL-Lite**

# TBox and ABox reasoning services

- **Ontology Satisfiability:** Verify whether an ontology  $\mathcal{O}$  is satisfiable, i.e., whether  $\mathcal{O}$  admits at least one model.
- **Concept Instance Checking:** Verify whether an individual  $c$  is an instance of a concept  $C$  in an ontology  $\mathcal{O}$ , i.e., whether  $\mathcal{O} \models C(c)$ .
- **Role Instance Checking:** Verify whether a pair  $(c, c')$  of individuals is an instance of a role  $R$  in an ontology  $\mathcal{O}$ , i.e., whether  $\mathcal{O} \models R(c, c')$ .
- **Query Answering** Given a query  $q$  over an ontology  $\mathcal{O}$ , find all tuples  $\bar{c}$  of constants such that  $\mathcal{O} \models q(\bar{c})$ .

# Query answering and instance checking

For **atomic concepts and roles**, **instance checking is a special case of query answering**, in which the query is **Boolean** and constituted by a single atom in the body.

- $O \models A(c)$       iff       $q = \{( \ ) | A(c)\}$  evaluated over  $O$  is *true*.
- $O \models P(c, c')$       iff       $q = \{( \ ) | P(c, c')\}$  evaluated over  $O$  is *true*.

# From instance checking to ontology unsatisfiability

**Theorem:** Let  $\mathcal{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  be a **DL-Lite** ontology,  $C$  a DL-Lite concept, and  $P$  an atomic role. Then:

- $\mathcal{O} \models C(c)$  iff  $\mathcal{O}_{C(c)} = \langle \mathbf{T} \cup \{\widehat{C} \sqsubseteq \neg C\}, \mathbf{A} \cup \{\widehat{C}(c)\} \rangle$  is **unsatisfiable**, where  $\widehat{C}$  is a new atomic concept not appearing in  $\mathcal{O}$ .
- $\mathcal{O} \models P(c, c')$  iff  $\mathcal{O}_{P(c, c')} = \langle \mathbf{T} \cup \{\widehat{P} \sqsubseteq \neg P\}, \mathbf{A} \cup \{\widehat{P}(c, c')\} \rangle$  is **unsatisfiable**, where  $\widehat{P}$  is a new atomic role not appearing in  $\mathcal{O}$ .

**Theorem:** Let  $\mathcal{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  be a **DL-Lite<sub>F</sub>** ontology and  $P$  an atomic role. Then  $\mathcal{O} \models P(c, c')$  iff the ontology  $\mathcal{O}$  is **unsatisfiable** or if  $P(c, c') \in \mathbf{A}$ .

## Certain answers

We recall that:

Query answering over an ontology  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  is a form of **logical implication**: find all tuples  $\bar{c}$  of constants such that  $\mathbf{O} \vDash q(\bar{c})$ .

a.k.a. **certain answers** in incomplete databases, i.e., the tuples that are answers to  $q$  in all models of  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$

$$cert(q, \mathbf{O}) = \{\bar{c} \mid \bar{c} \in q^I \text{ for every models } I \text{ of } \mathbf{O}\}$$

**Note:** we assume that the answer  $q^I$  to a query  $q$  over an interpretation  $I$  is constituted by a set of tuples of constants of  $\mathbf{A}$ , rather than objects in  $\Delta^I$ .

## FOL-rewritability for DL-Lite

We now study **rewritability** of query answering over **DL-Lite** ontologies.

In particular we will show that **DL-Lite<sub>A</sub>** (and hence **DL-Lite<sub>F</sub>** and **DL-Lite<sub>R</sub>**) enjoy **FOL-rewritability** of answering union of conjunctive queries (UCQ).

That is:

Given an ontology **O** in **DL-Lite** answering UCQ is **FOL-rewritable**, i.e., for every TBox **T** in **DL-Lite** and for every UCQ **q**, the perfect reformulation  $r_{q,T}$  of **q** w.r.t. **T** can be expressed as a query in **FOL** (and so in **SQL**).

# Query answering vs. ontology satisfiability

In the case in which an ontology is **unsatisfiable**, according to the "*ex falso quodlibet*" principle, **reasoning is trivialized**.

In particular, **query answering is meaningless**, since every tuple is in the answer to every query.

Clearly, we are **not interested** in encoding meaningless query answering into the perfect reformulation of the input query. Therefore, before query answering, we will always check ontology satisfiability to single out meaningful cases.

Thus, we proceed as follows:

1. We show how to do **query answering over satisfiable ontologies**.
2. We show how we can exploit the query answering algorithm also to **check ontology satisfiability**.

## Positive vs. negative inclusions

We call **positive inclusions (PIs)** assertions of the form

$$A_1 \sqsubseteq A_2$$

$$\exists Q \sqsubseteq A$$

$$Q_1 \sqsubseteq Q_2$$

$$A \sqsubseteq \exists Q$$

$$\exists Q_1 \sqsubseteq \exists Q_2$$

Given a **TBox T**, we denote by **T<sub>P</sub>** the subset of **T** containing only the **PIs** in **T**.

We call **negative inclusions (NIs)** assertions of the form

$$A_1 \sqsubseteq \neg A_2$$

$$\exists Q \sqsubseteq \neg A$$

$$Q_1 \sqsubseteq \neg Q_2$$

$$A \sqsubseteq \neg \exists Q$$

$$\exists Q_1 \sqsubseteq \exists \neg Q_2$$

# Query answering over satisfiable ontologies

Given a CQ  $q$  and a **satisfiable** ontology  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  we compute  $\text{cert}(q, \mathbf{O})$  as follows:

1. By using the TBox  $\mathbf{T}$  (only), we **rewrite**  $q$  into a UCQ  $r_{q,T}$  (the **perfect rewriting** of  $q$  w.r.t.  $\mathbf{T}$ ).
2. We **evaluate**  $r_{q,T}$  over  $\mathbf{A}$  (simply viewed as data), to return  $\text{cert}(q, \mathbf{O})$

Correctness of this procedure shows **FOL-rewritability** of query answering in DL-Lite.

# Query rewriting step: Basic idea

**Intuition:** a **positive inclusion** in the TBox corresponds to a **logic programming rule**.

**Basic rewriting step:** When an atom in the query unifies with the head of the rule, generate a new query by substituting the atom with the body of the rule. In this case, we say that the positive inclusion **applies** to the atom.

**Example:** The positive inclusion **AssistantProf  $\sqsubseteq$  Professor** corresponds to the logic programming rule **Professor(z)  $\leftarrow$  AssistantProf(z)**.

Consider the input query  **$q = \{(x) \mid \text{Professor}(x)\}$** .

By applying the positive inclusion to the atom **Professor(x)** in the query, we generate the new query:

- **$q_1 = \{(x) \mid \text{AssistantProf}(x)\}$** .

This query is **added** to the input query, and contributes to the perfect rewriting.

## Query rewriting – Example 1

Consider the input query

$$q = \{(x) \mid \exists y. \text{teaches}(x, y) \wedge \text{Course}(y)\}$$

and the PI:  $\exists y. \text{teaches}(x, y) \sqsubseteq \text{Course}(y)$  as the logic programming rule:

$$\text{Course}(z_2) \leftarrow \text{teaches}(z_1, z_2)$$

The PI applies to the atom  $\text{Course}(y)$  of the input query, so we add to the perfect rewriting the query:

$$q' = \{(x) \mid \exists y, z_1. \text{teaches}(x, y) \wedge \text{teaches}(z_1, y)\}$$

## Query rewriting – Example 2

Consider the input query

$$q = \{(x) \mid \exists y.\text{teaches}(x, y)\}$$

and the PI: **Professor**  $\sqsubseteq \exists \text{teaches}$  as the logic programming rule:

$$\text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)$$

Where  $f(x)$  is a **Skolem term**.

The PI applies to the atom **teaches**( $x, y$ ) of the input query, so we add to the perfect rewriting the query:

$$q' = \{(x) \mid \text{Professor}(x)\}$$

## Query rewriting – Example 3

Consider the input query

$$q = \{(x) \mid \exists y. \text{teaches}(x, \text{database})\}$$

and the same PI: **Professor**  $\sqsubseteq \exists \text{teaches}$  as the logic programming rule:

$$\text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)$$

The atom **teaches(x, database)** **does not unify** with **teaches(z, f(z))** since the **Skolem term** **f(z)** in the head of the rule **does not unify** with the constant **database**

In this case, we say that the PI **does not apply** to the atom **teaches(x, database)**, so, in this case, we do not add any new query to the perfect rewriting.

## Query rewriting – Example 4

Consider the input query (note that  $y$  is now a **distinguished variable**)

$$q = \{(x, y) \mid \text{teaches}(x, y)\}$$

and the same PI: **Professor**  $\sqsubseteq \exists \text{teaches}$  as the logic programming rule:

$$\text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)$$

Also in this case we have **no unification**. Indeed unifying the **Skolem term**  $f(z)$  the variable  $y$  would correspond to returning a **Skolem term** as answer to the input query.

## Query rewriting – Example 5 (join variables)

An analogous behavior to the one with **constants** and with **distinguished variables** holds when the atom contains **join variables** that would have to be unified with Skolem terms.

Consider the input query:

$$q = \{(x) \mid \exists y. \text{teaches}(x, y) \wedge \text{Course}(y)\}$$

and the same PI: **Professor**  $\sqsubseteq \exists \text{teaches}$  as the logic programming rule:

$$\text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)$$

The PI above does **not apply** to the atom **teaches(x, y)**.

## Query rewriting – The Reduce step

Consider the query:  $q' = \{(x) \mid \exists y, z_1. \text{teaches}(x, y) \wedge \text{teaches}(z_1, y)\}$   
and the same PI: **Professor**  $\sqsubseteq \exists \text{teaches}$  as the logic programming rule:

$$\text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)$$

As we saw, the PI above does not apply to  $\text{teaches}(x, y)$  or  $\text{teaches}(x, y)$ , since  $y$  is in **join**, and we would again introduce the **Skolem term** in the rewritten query. However, we can transform the above query by **unifying** the atoms  $\text{teaches}(x, y)$  and  $\text{teaches}(z, y)$ . This rewriting step is called **reduce**, and produces the query

$$q' = \{(x) \mid \text{teaches}(x, y)\}$$

Now, we can apply the PI above, and add to the rewriting the query

$$q'' = \{(x) \mid \text{Professor}(x)\}$$

# Query rewriting – the procedure (1\2)

To compute the perfect rewriting of a UCQ  $q$ , start from  $q$ , iteratively get a CQ  $q'$  to be processed, and do one of the following steps:

- Apply to some atom of  $q'$  a PI in  $\mathbf{T}$  as follows:

|                                       |                            |                    |                            |
|---------------------------------------|----------------------------|--------------------|----------------------------|
| $A_1 \sqsubseteq A_2$                 | $\dots, A_2(x), \dots$     | $\rightsquigarrow$ | $\dots, A_1(x), \dots$     |
| $\exists P \sqsubseteq A$             | $\dots, A(x), \dots$       | $\rightsquigarrow$ | $\dots, P(x, \_), \dots$   |
| $\exists P^- \sqsubseteq A$           | $\dots, A(x), \dots$       | $\rightsquigarrow$ | $\dots, P(\_, x), \dots$   |
| $A \sqsubseteq \exists P$             | $\dots, P(x, \_), \dots$   | $\rightsquigarrow$ | $\dots, A(x), \dots$       |
| $A \sqsubseteq \exists P^-$           | $\dots, P(\_, x), \dots$   | $\rightsquigarrow$ | $\dots, A(x), \dots$       |
| $\exists P_1 \sqsubseteq \exists P_2$ | $\dots, P_2(x, \_), \dots$ | $\rightsquigarrow$ | $\dots, P_1(x, \_), \dots$ |
| $P_1 \sqsubseteq P_2$                 | $\dots, P_2(x, y), \dots$  | $\rightsquigarrow$ | $\dots, P_1(x, y), \dots$  |
| $P_1 \sqsubseteq P_2^-$               | $\dots, P_2(x, y), \dots$  | $\rightsquigarrow$ | $\dots, P_1(y, x), \dots$  |
|                                       |                            | :                  |                            |

where '\_' denotes an **unbound** variable, i.e., a variable that appears only once

continued on next slide 

## Query rewriting – the procedure (2\2)

...

- Find (if there are) two atoms of  $q'$  that **unify**, and apply the unifier to  $q'$ .

Each time, the result of the above steps is added to the queries to be processed.

**Note:** Unifying atoms can make rules applicable that were not so before, and is required for completeness of the method.

The process **stops** when neither of the previous two steps are applicable

The **UCQ** resulting from this process is the **perfect rewriting**  $r_{q,T}$  of the input query w.r.t. the **TBox**

# Query rewriting algorithm

**Algorithm**  $\text{PerfectRef}(Q; T_P)$

**Input:** union of conjunctive queries  $Q$ , set of DL-Lite<sub>A</sub> PIs  $T_P$

**Output:** union of conjunctive queries  $PR$

$PR := Q;$

**repeat**

$PR' := PR;$

**for each**  $q \in PR'$  **do**

**for each atom**  $g \in q$  **do**

**for each PI**  $I \in T_P$  **do**

**if**  $I$  is applicable to  $g$  **then**  $PR := PR \cup \{ \text{ApplyPI}(q, g, I) \};$

**for each pair of atoms**  $g_1, g_2 \in q$  **do**

**if**  $g_1$  and  $g_2$  unify **then**  $PR := PR \cup \{ \text{Reduce}(q, g_1, g_2) \};$

**until**  $PR' = PR;$

**return**  $PR;$

**Observations:** 1) **Termination** follows from having only finitely many different rewritings.  
2) **NIs** or **functionalities** do not play **any role** in the rewriting of the query.

# Perfect rewriting in DL-Lite – Example

$T = \{ \text{AssistantProf} \sqsubseteq \text{Professor},$   
 $\text{Professor} \sqsubseteq \exists \text{teaches},$   
 $\exists \text{teaches}^- \sqsubseteq \text{Course} \}$

$A = \{ \text{teaches(john, math)}, \text{AssistantProf(john)},$   
 $\text{teaches(bob, database)}, \text{AssistantProf(mary)} \}$

**Query:**  $q : \{(x) \mid \exists y. \text{teaches}(x, y) \wedge \text{Course}(y)\}$

**Perfect rewriting:**  $r_{q,T} :$

$$\begin{aligned} & \{(x) \mid \exists y. \text{teaches}(x, y) \wedge \text{Course}(y)\} \\ & \{(x) \mid \exists y, s. \text{teaches}(x, y) \wedge \text{teaches}(s, y)\} \\ & \{(x) \mid \exists y. \text{teaches}(x, y)\} \\ & \{(x) \mid \text{Professor}(x)\} \\ & \{(x) \mid \text{AssistantProf}(x)\} \end{aligned}$$

Evaluating  $r_{q,T}$  over the **ABox A** (seen as a DB) produces as **answer** {john, bob, mary}.

# Query answering over satisfiable DL-Lite ontologies

For an **ABox**  $\mathbf{A}$  and a query  $q$  over  $\mathbf{A}$ , let  $\text{Eval}_{\text{cwa}}(q, \mathbf{A})$  denote the evaluation of  $q$  over  $\mathbf{A}$  considered as a **database** (i.e., considered under the **CWA**).

**Theorem:** Let  $\mathbf{T}$  be a **DL-Lite TBox**,  $\mathbf{T}_P$  the set of **PIs** in  $\mathbf{T}$ , and  $q$  a **CQ** over  $\mathbf{T}$ . Then, for each **ABox**  $\mathbf{A}$  such that  $\langle \mathbf{T}, \mathbf{A} \rangle$  is **satisfiable**, we have that

$$\text{cert}(q, \langle \mathbf{T}, \mathbf{A} \rangle) = \text{Eval}_{\text{cwa}}(\text{PerfectRef}(q, \mathbf{T}_P), \mathbf{A}):$$

As a consequence, query answering over a satisfiable **DL-Lite** ontology is **FOL-rewritable**.

Notice that we did not use **NIs** or **functionality** assertions of  $\mathbf{T}$  in computing  $\text{cert}(q, \langle \mathbf{T}, \mathbf{A} \rangle)$ . Indeed, when the ontology is satisfiable, we can ignore Nis and functionality assertions for query answering.

# Using RDBMS technology for query answering

The **ABox A** can be stored as a **relational database** in a standard RDBMS:

- For each **atomic concept C** of the ontology define a **unary relational table tabC**, and populate it with each  $\langle c \rangle$  such that  $C(c) \in A$ .
- For each **atomic role P** of the ontology, define a **binary relational table tabP**, and populate it with each  $\langle c_1, c_2 \rangle$  such that  $P(c_1, c_2) \in A$ .

We have that query answering over satisfiable DL-Lite ontologies can be done effectively using RDBMS technology:

$$\text{cert}(q, \langle T, A \rangle) = \text{Eval}(\text{SQL}(\text{PerfectRef}(q, T_P)), DB(A))$$

Where:

- **Eval(q,DB)** denotes the evaluation of an SQL query **q** over a database **DB**.
- **SQL(q)** denotes the SQL encoding of a UCQ **q**.
- **DB(A)** denotes the database obtained as above.

# Satisfiability of ontologies with only Positive Inclusions

Let us now consider the problem of establishing whether an ontology is satisfiable.

A first notable result tells us that positive inclusion assertions alone cannot give rise to unsatisfiable ontologies.

**Theorem:** Let  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  be a DL-Lite ontology where  $\mathbf{T}$  contains **only PIs**.  
Then,  $\mathbf{O}$  is satisfiable.

→ Unsatisfiability in  $\text{DL-Lite}_A$  ontologies can be caused by **NIs** or by **functionality assertions**.

# Checking satisfiability of DL-Lite<sub>A</sub> ontologies

Satisfiability of a DL-Lite<sub>A</sub> ontology  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  is reduced to evaluating over  $\mathbf{DB}(\mathbf{A})$  a query that asks for the existence of objects violating the **NIs** or **functionality** assertions.

Let  $\mathbf{T}_P$  be the set of **PIs** in  $\mathbf{T}$ . We deal with **NIs** and **functionality** assertions differently

- For each **NI**  $N \in \mathbf{T}$ , we construct a Boolean CQ  $\mathbf{q}_N$  such that  $\langle \mathbf{T}_P, \mathbf{A} \rangle \models \mathbf{q}_N$  iff  $\langle \mathbf{T}_P \cup \{N\}, \mathbf{A} \rangle$  is **unsatisfiable**.
- We check whether  $\langle \mathbf{T}_P, \mathbf{A} \rangle \models \mathbf{q}_N$  by using **PerfectRef**, i.e., we compute  $\mathbf{PerfectRef}(\mathbf{q}_N, \mathbf{T}_P)$ , and evaluate it over  $\mathbf{A}$  (considered as a database).
- For each **functionality assertion**  $F \in \mathbf{T}$ , we construct a Boolean FOL query  $\mathbf{q}_F$  such that  $\mathbf{A} \models \mathbf{q}_F$ , iff  $\langle \{F\}, \mathbf{A} \rangle$  is **unsatisfiable**.
- We check whether  $\mathbf{A} \models \mathbf{q}_F$  by directly evaluating  $\mathbf{q}_F$  over  $\mathbf{A}$  (considered as a database).

# Checking violations of negative inclusions

For each NI  $N$  in  $\mathbf{T}$  we compute a boolean CQ  $q_N$  according to the following rules:

| $N$                                                                  |   | $q_N$                                                    |
|----------------------------------------------------------------------|---|----------------------------------------------------------|
| $A_1 \sqsubseteq \neg A_2$                                           | ⇒ | $\{(\ )   \exists x. A_1(x) \wedge A_2(x)\}$             |
| $\exists P \sqsubseteq \neg A$ or $A \sqsubseteq \neg \exists P$     | ⇒ | $\{(\ )   \exists x, y. P(x, y) \wedge A(x)\}$           |
| $\exists P^- \sqsubseteq \neg A$ or $A \sqsubseteq \neg \exists P^-$ | ⇒ | $\{(\ )   \exists x, y. P(x, y) \wedge A(y)\}$           |
| $\exists P_1 \sqsubseteq \neg \exists P_2$                           | ⇒ | $\{(\ )   \exists x, y, z. P_1(x, y) \wedge P_2(x, z)\}$ |
| $\exists P_1 \sqsubseteq \neg \exists P^-_2$                         | ⇒ | $\{(\ )   \exists x, y, z. P_1(x, y) \wedge P_2(z, x)\}$ |
| $\exists P^-_1 \sqsubseteq \neg \exists P_2$                         | ⇒ | $\{(\ )   \exists x, y, z. P_1(x, y) \wedge P_2(y, z)\}$ |
| $\exists P^-_1 \sqsubseteq \neg \exists P^-_2$                       | ⇒ | $\{(\ )   \exists x, y, z. P_1(x, y) \wedge P_2(z, y)\}$ |
| $P_1 \sqsubseteq \neg P_2$ or $P^-_1 \sqsubseteq \neg P^-_2$         | ⇒ | $\{(\ )   \exists x, y. P_1(x, y) \wedge P_2(x, y)\}$    |
| $P^-_1 \sqsubseteq \neg P_2$ or $P_1 \sqsubseteq \neg P^-_2$         | ⇒ | $\{(\ )   \exists x, y. P_1(x, y) \wedge P_2(y, x)\}$    |

# Checking violations of functionality assertions

For each **functionality assertions  $F$**  in  $\mathbf{T}$  we compute a Boolean FOL query  $q_F$  according to the following rules:

***functionality assertion***

***q<sub>F</sub>***

**(funct P)**

$\Rightarrow \{(\quad) | \exists x, y, z. P_1(x, y) \wedge P_2(x, z) \wedge y \neq z\}$

**(funct P<sup>-</sup>)**

$\Rightarrow \{(\quad) | \exists x, y, z. P_1(y, x) \wedge P_2(z, x) \wedge y \neq z\}$

# From satisfiability to query answering in DL-Lite<sub>A</sub>

**Lemma (Separation for DL-Lite<sub>A</sub>):** Let  $\mathcal{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  be a DL-Lite<sub>A</sub> ontology, and  $\mathbf{T}_P$  the set of PIs in  $\mathbf{T}$ . Then,  $\mathcal{O}$  is **unsatisfiable** iff one of the following condition holds:

- (a) There exists a NI  $N \in \mathbf{T}$  such that  $\langle \mathbf{T}, \mathbf{A} \rangle \models q_N$
- (b) There exists a functionality assertion  $F \in \mathbf{T}$  such that  $\mathbf{A} \models q_F$ .

(a) relies on the properties that NIs do not interact with each other, and that interaction between NIs and PIs is captured through **PerfectRef**.

(b) exploits the property that NIs and PIs do not interact with functionalities: indeed, no functionality assertion is contradicted in a DL-Lite<sub>A</sub> ontology, beyond those explicitly contradicted by the **ABox**.

Notably, to check ontology satisfiability, each NI and each functionality assertion can be processed individually.

## FOL-rewritability of satisfiability in DL-Lite<sub>A</sub>

From the previous lemma and the theorem on query answering for satisfiable **DL-Lite<sub>A</sub>** ontologies, we get the following result.

**Theorem:** Let  $\mathbf{O} = \langle \mathbf{T}, \mathbf{A} \rangle$  be a **DL-Lite<sub>A</sub>** ontology, and  $\mathbf{T}_P$  the set of PIs in  $\mathbf{T}$ . Then,  $\mathbf{O}$  is **unsatisfiable** iff one of the following condition holds:

- (a) There exists a NI  $N \in \mathbf{T}$  s.t.  $\text{Eval}_{\text{cwa}}(\text{PerfectRef}(q_N, \mathbf{T}_P), \mathbf{A})$  returns **true**.
- (b) There exists a func. assertion  $F \in \mathbf{T}$  s.t.  $\text{Eval}_{\text{cwa}}(q_F, \mathbf{A})$  returns **true**.

**Note:** All the queries  $q_N$  and  $q_F$  can be combined into a single UCQ. Hence, satisfiability of a **DL-Lite<sub>A</sub>** ontology is reduced to evaluating a FOL-query over an ontology whose TBox consists of positive inclusions only (and hence is satisfiable).

# Complexity of reasoning in DL-Lite<sub>A</sub>

**Theorem:** **Query answering** over DL-Lite<sub>A</sub> ontologies is

- **NP-complete** in the size of **query and ontology** (combined complexity).
- **PTime** in the size of the **ontology** (schema + data complexity).
- **AC<sup>0</sup>** in the size of the **ABox** (data complexity).

**Theorem:** **Checking satisfiability** of DL-Lite<sub>A</sub> ontologies is

- **PTime** in the size of the **ontology** (schema + data complexity).
- **AC<sup>0</sup>** in the size of the **ABox** (data complexity).

**Theorem:** **TBox reasoning** over DL-Lite<sub>A</sub> ontologies is **PTime** in the size of the **TBox** (schema complexity).

# **Linking ontologies to relational data (Query answering in OBDM)**

# Managing ABoxes

In the traditional DL setting, it is assumed that the data is maintained in the **ABox** of the ontology:

- The **ABox** is perfectly compatible with the **TBox**:
  - the vocabulary of concepts, roles, and attributes is the one used in the **TBox**.
  - the **ABox** "stores" abstract objects, and these objects and their properties are those returned by queries over the ontology.
- There may be different ways to manage the **ABox** from a physical point of view:
  - DLs reasoners maintain the **ABox** in main-memory data structures.
  - when an **ABox** becomes large, managing it in secondary storage may be required, but this is again handled directly by the reasoner.

# Data in external sources

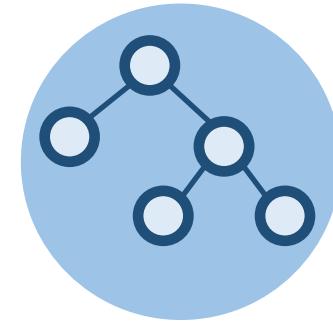
There are several situations where the assumptions of having the data in an **ABox** managed directly by the ontology system (e.g., a Description Logics reasoner) is not feasible or realistic:

- When the **ABox** is very large, so that it requires relational database technology.
- When we have no direct control over the data since it belongs to some external organization, which controls the access to it.
- When **multiple data sources** need to be accessed, such as in Information Integration.
- We would like to deal with such a situation by keeping the data in the external (relational) storage, and performing **query answering** by leveraging the capabilities of the **relational engine**.

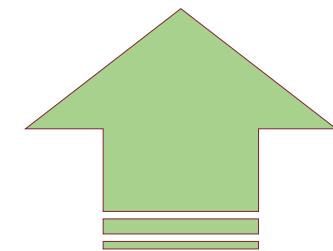
# Ontology-based data management: Architecture

The architecture of an **OBDM** system is based on three main components:

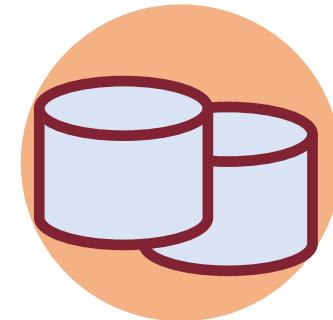
- **Ontology**: provides a unified, conceptual view of the managed information.
- **Data source(s)**: are external and independent (possibly multiple and heterogeneous).
- **Mappings**: semantically link data at the sources with the ontology.



Ontology



Mapping



Data sources  
(source schema)

# The impedance mismatch problem

We have to deal with the **impedance mismatch problem**:

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, . . .
- Instead, instances of concepts and relations in an ontology are (abstract) objects.

## Solution:

- We need to specify how to construct from the data values in the relational sources the (abstract) objects that populate the **ABox** of the ontology.
- This specification is embedded in the mappings between the data sources and the ontology.

**Note:** the **ABox** here is only **virtual**, and the objects are not materialized.

# Solution to the impedance mismatch problem

We need to define a **mapping language** that allows for specifying how to transform data into abstract objects:

- Each mapping assertion maps:
  - a query that retrieves values from a data source to . . .
  - a set of atoms specified over the ontology.
- **Basic idea:** We need **constructors** to "create" ontology objects out of tuples of values in the database. Formally, such constructors can be **Skolem functions**.
- **Semantics of mappings:**
  - Objects are denoted by terms.
  - Different terms denote different objects (i.e., we make the **unique name assumption** on terms).

## Impedance mismatch – Example

Suppose our data are stored in a database containing information about employees and the projects they work in.

- D1[SSN:String; PrName:String] : Employees and projects they work for
- D2[Code:String; Salary:Int] : Employee's code with salary
- D3[Code:String; SSN:String] : Employee's Code with SSN

We know that:

- An employee is **identified** by her **SSN**.
- A project is **identified** by its **name**.

**Intuitively:**

- An object for an employee should be created from her SSN: **pers(SSN)**
- An objects for a project should be created from its name: **proj(PrName)**

# Impedance mismatch: the technical solution

We need to associate the data in the tables to objects in the ontology.

- We introduce an alphabet  $\Lambda$  of **function symbols**, each with an associated arity.
- Let  $\Gamma_V$  be the alphabet of constants (values) appearing in the sources.
- To denote objects, i.e., instance of the concepts in the ontology, we use **object terms** instead of **object constants**.
- An **object term** has the form  $f(d_1, \dots, d_n)$ , with  $f \in \Lambda$ , and each  $d_i$  a value constant in  $\Gamma_V$ .

## Example

- If a person is identified by her **SSN**, we can introduce a function symbol  $\text{pers}_{/1}$ . If **VRD56B25** is a **SSN**, then  $\text{pers}(\text{VRD56B25})$  denotes a person.
- If a person is identified by her **name** and **dateOfBirth**, we can introduce a function symbol  $\text{pers}_{/2}$ . Then  $\text{pers}(\text{Bob}, \text{24/05/1941})$  denotes a person.

# Let's pick up from where we left off

In the context of ontology-based data management:

- Which is the "right" (user's) **query language** → UCQ
- Which is the "right" **ontology language** → DL-Lite
- How can we bridge the **semantic mismatch** between the ontology and the data sources? → We use **object terms** instead of **object constants**.  
To generate these terms we make use of **Skolem functions**.

# Mapping assertions

Mapping assertions are used to extract the data from the DB to populate the ontology.

We make use of **variable terms**, which are like **object terms**, but with **variables** instead of **values** as arguments of the functions.

**Definition:** A **mapping assertion** between a database **D** and a TBox **T** has the form\*

$$\Phi(\bar{x}) \rightsquigarrow \Psi(\bar{t}, \bar{y})$$

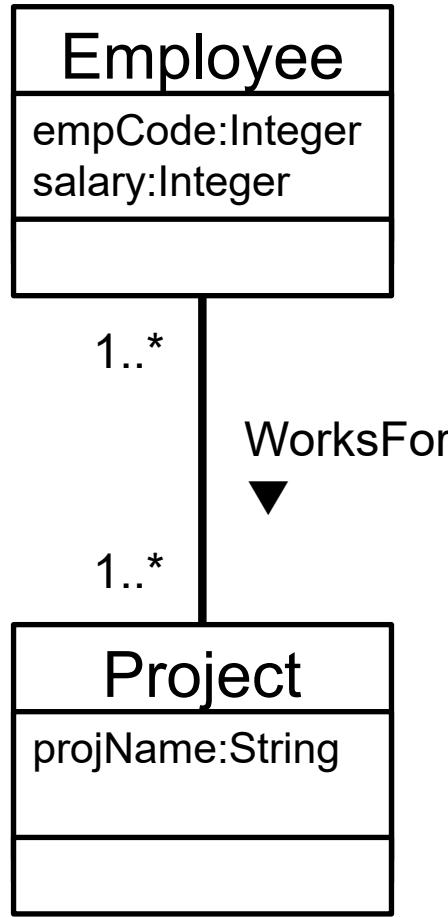
where:

- $\Phi(\bar{x})$  is an arbitrary SQL query of arity  $n > 0$  over **D**;
- $\Psi(\bar{t}, \bar{y})$  is a conjunction of atoms whose predicates are **atomic concepts** and **roles** of **T**;
- $\bar{x}$  and  $\bar{y}$  are variables, with  $\bar{x} \subseteq \bar{y}$
- $\bar{t}$  are **variable terms** of the form  $f(\bar{z})$ , with  $f \in \Lambda$  and  $\bar{z} \subseteq \bar{x}$ .

\* Note: this is a form of GAV mapping

# Mapping assertions – Example (1/2)

Consider the following TBox T (UML)

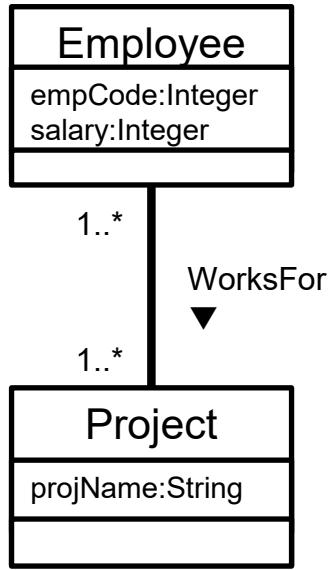


**DL-Lite<sub>A</sub> axioms**

$\text{Employee} \sqsubseteq \exists \text{WorksFor}$   
 $\text{Project} \sqsubseteq \exists \text{WorksFor}^-$   
 $\exists \text{WorksFor} \sqsubseteq \text{Employee}$   
 $\exists \text{WorksFor}^- \sqsubseteq \text{Project}$   
 $\text{Employee} \sqsubseteq \delta(\text{empCode})$   
 $\text{Employee} \sqsubseteq \delta(\text{salary})$   
 $\text{Project} \sqsubseteq \delta(\text{projName})$   
 $\delta(\text{empCode}) \sqsubseteq \text{Employee}$   
 $\delta(\text{empCode}) \sqsubseteq \text{Employee}$   
 $\delta(\text{projName}) \sqsubseteq \text{Project}$   
 $\rho(\text{empCode}) \sqsubseteq \text{xsd:integer}$   
 $\rho(\text{salary}) \sqsubseteq \text{xsd:integer}$   
 $\rho(\text{projName}) \sqsubseteq \text{xsd:string}$   
 $(\text{funct } \text{empCode})$   
 $(\text{funct } \text{salary})$   
 $(\text{funct } \text{projName})$

# Mapping assertions – Example (2/2)

## TBox T (UML)



## Source schema S

D1(SSN:string, PrName:string)

Employees and the project they work for

D2(Code:string, Salary:integer)

Employee's Code with salary

D3(SSN:string, Code:integer)

Employee's Code with SSN

...

## Mapping M

m<sub>1</sub>: `SELECT SSN, PrName  
FROM D1`



Employee(**pers(SSN)**),  
Project(**prj(PrName)**),  
WorksFor(**pers(SSN)**, **prj(PrName)**),  
projName(**prj(PrName)**, PrName)

m<sub>2</sub>: `SELECT SSN, Code  
FROM D3`



Employee(**pers(SSN)**),  
empCode(**pers(SSN)**, Code)

m<sub>3</sub>: `SELECT SSN, Salary  
FROM D2, D3  
WHERE D2.Code = D3.Code`



Employee(**pers(SSN)**),  
Salary(**pers(SSN)**, Salary)

# Ontology-Based Data Management: Formalization

To formalize OBDM, we distinguish between the **intensional** and the **extensional** level information.

**Definition:** An **OBDM specification** is a triple  $P = \langle T, M, S \rangle$ , where:

- $T$  is a **DL TBox** providing the intensional level of an ontology.
- $S$  is a (possibly federated) relational database schema for the **data sources**, possibly with constraints;
- $M$  is a set of **mapping assertions** between  $T$  and  $S$ .

**Definition:** An **OBDM instance (or system)** is a pair  $O = \langle P, D \rangle$ , where:

- $P = \langle T, M, S \rangle$  is an **OBDM specification**, and
- $D$  is a **relational database** for  $S$ .

# Semantics of an OBDM instance: Intuition

Given an **OBDM instance**, the **mapping M** encodes how the data **D** in the source(s) **S** should be used to populate the elements of the **TBox T**.

The data **D** and the mapping **M** define a **virtual data layer V**, which behaves like a **(virtual) ABox**.

Queries are answered w.r.t. **T** and **V**.

→ We want to **avoid materializing** the data of **V**. Instead, the intensional information in **T** and **M** is used to **translate** queries over **T** into queries formulated over **S**.

## OBDM vs. Query Answering in Ontologies (QAO)

- **OBDM** relies on **QAO** to process queries w.r.t. the **TBox T**, but in addition is concerned with **efficiently** dealing with the **mapping M**.

# Semantics of mappings

To define the semantics of an OBDM instance  $\mathbf{O} = \langle \mathbf{P}, \mathbf{D} \rangle$  with  $\mathbf{P} = \langle \mathbf{T}, \mathbf{M}, \mathbf{S} \rangle$ , we first need to define the semantics of mappings.

## Definition: Satisfaction of a mapping assertion with respect to a database

An interpretation  $I$  satisfies a mapping assertion  $\Phi(\bar{x}) \rightsquigarrow \Psi(\bar{t}, \bar{y})$  in  $\mathbf{M}$  with respect to a database  $\mathbf{D}$ , if for each tuple of values  $\bar{v} \in \text{Eval}(\Phi, \mathbf{D})$ , and for each ground atom in  $\Psi[\bar{x} \setminus \bar{v}]$ , we have that:

- if the ground atom is  $A(s)$ , then  $s^I \in A^I$ .
- if the ground atom is  $P(s_1, s_2)$ , then  $(s_1,^I s_2^I) \in P^I$

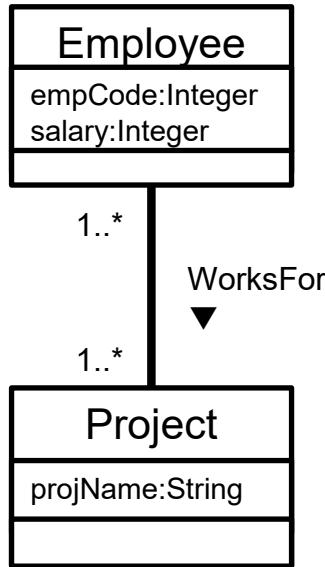
Intuitively,  $I$  satisfies  $\Phi \rightsquigarrow \Psi$  w.r.t.  $\mathbf{D}$  if all facts obtained by evaluating  $\Phi$  over  $\mathbf{D}$  and then propagating the answers to  $\Psi$ , hold in  $I$ .

**Note:**  $\text{Eval}(\Phi, \mathbf{D})$  denotes the result of evaluating  $\Phi$  over the database  $\mathbf{D}$ .

$\Psi[\bar{x} \setminus \bar{v}]$  denotes  $\Psi$  where each  $x_i$  has been substituted with  $v_i$ .

# Semantics of mappings – Example

## TBox T (UML)



## Source database D

| $D_1$ |        |
|-------|--------|
| SSN   | PrName |
| 23AB  | Apollo |

| $D_2$ |        |
|-------|--------|
| Code  | Salary |
| e23   | 15000  |

| $D_3$ |      |
|-------|------|
| SSN   | Code |
| 23AB  | e23  |

## Mapping M

$m_1:$  `SELECT SSN, PrName  
FROM D1`

$\rightsquigarrow$   
 $\text{Employee}(\mathbf{pers(SSN)}),$   
 $\text{Project}(\mathbf{prj(PrName)}),$   
 $\text{WorksFor}(\mathbf{pers(SSN)}, \mathbf{prj(PrName)}),$   
 $\text{projName}(\mathbf{prj(PrName)}, \mathbf{PrName})$

The following interpretation satisfies  $m_1$  with respect to  $D$  (note that we are directly using **object terms** as domain elements):

$I : \Delta_O^I = \{\mathbf{pers}(23AB), \mathbf{prj}(Apollo), \dots\}, \Delta_V^I = \{Apollo, 15000, \dots\},$   
 $\text{Employee}^I = \{\mathbf{pers}(23AB), \dots\},$   
 $\text{Project}^I = \{\mathbf{prj}(Apollo), \dots\},$   
 $\text{WorksFor}^I = \{(\mathbf{pers}(23AB), \mathbf{prj}(Apollo)), \dots\},$   
 $\text{projName}^I = \{(\mathbf{prj}(Apollo), \mathbf{Apollo}), \dots\}\}, \dots$

# Semantics of an OBDM instance

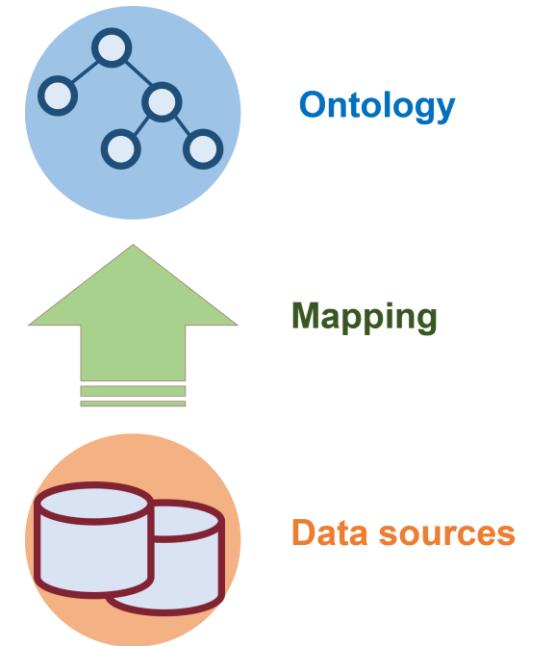
## Definition: Model of an OBDM instance

An interpretation  $I$  is a **model** of an OBDM instance

$\mathbf{O} = \langle \mathbf{P}, \mathbf{D} \rangle$  with  $\mathbf{P} = \langle \mathbf{T}, \mathbf{M}, \mathbf{S} \rangle$ , if:

- $I$  is a model of  $\mathbf{T}$ , and
- $I$  satisfies  $\mathbf{M}$  w.r.t.  $\mathbf{D}$ , i.e.,  $I$  satisfies every mapping assertion in  $\mathbf{M}$  w.r.t.  $\mathbf{D}$ .

An OBDM instance  $\mathbf{O}$  is **satisfiable** if it admits at least one **model**.



# Answering query over an OBDM system

Consider an OBDM instance  $\mathbf{O} = \langle \mathbf{P}, \mathbf{D} \rangle$  with  $\mathbf{P} = \langle \textcolor{blue}{\mathbf{T}}, \textcolor{green}{\mathbf{M}}, \textcolor{brown}{\mathbf{S}} \rangle$ :

- Queries are posed over the TBox  $\textcolor{blue}{\mathbf{T}}$ .
- The data needed to answer queries is stored in the database  $\mathbf{D}$ , which is compliant to  $\textcolor{brown}{\mathbf{S}}$ .
- The mapping  $\mathbf{M}$  is used to bridge the gap between  $\textcolor{blue}{\mathbf{T}}$  and  $\mathbf{D}$ .

**Two approaches are possible:**

- **bottom-up approach** (materialization approach): simpler, but typically less efficient
- **top-down approach** (virtualization approach): more sophisticated, but also more efficient

Both approaches require to first **split** the TBox queries in the mapping assertions into their constituent atoms (in order to obtain "**pure**" GAV Mapping).

# Splitting of Mapping

A mapping assertions  $\Phi(\bar{x}) \rightsquigarrow \Psi(\bar{t}, \bar{y})$  where the query  $\Psi(\bar{t}, \bar{y})$  is constituted by the atoms  $\alpha_1, \dots, \alpha_k$ , can be split into several mapping assertions:

$$\Phi \rightsquigarrow \alpha_1, \dots, \Phi \rightsquigarrow \alpha_k$$

This can be done, since  $\Psi$  does **not contain existential variables**

**Example:**

$m_1: \text{SELECT SSN, PrName FROM D1} \rightsquigarrow \text{Employee}(\text{pers}(SSN)), \text{Project}(\text{prj}(PrName)), \text{WorksFor}(\text{pers}(SSN), \text{prj}(PrName)), \text{projName}(\text{prj}(PrName), PrName)$

is splitted into:

$m_{1-1}: \text{SELECT SSN, PrName FROM D1} \rightsquigarrow \text{Employee}(\text{pers}(SSN))$

$m_{1-2}: \text{SELECT SSN, PrName FROM D1} \rightsquigarrow \text{Project}(\text{prj}(PrName))$

$m_{1-3}: \text{SELECT SSN, PrName FROM D1} \rightsquigarrow \text{WorksFor}(\text{pers}(SSN), \text{prj}(PrName))$

$m_{1-4}: \text{SELECT SSN, PrName FROM D1} \rightsquigarrow \text{projName}(\text{prj}(PrName), PrName)$

# Bottom-up approach to query answering

Consider an OBDM instance  $\mathbf{O} = \langle \mathbf{P}, \mathbf{D} \rangle$  with  $\mathbf{P} = \langle \mathbf{T}, \mathbf{M}, \mathbf{S} \rangle$ :

The approach consists in a straightforward application of the mappings in order to compute a **materialized ABox**:

- Propagate the data from  $\mathbf{D}$  through  $\mathbf{M}$ , **materializing** an ABox  $A_{\mathbf{M}, \mathbf{D}}$  (the constants in such an ABox are **values** and **object terms**).
- Apply to  $A_{\mathbf{M}, \mathbf{D}}$  and to the TBox  $\mathbf{T}$ , the **satisfiability** and **query answering** algorithms developed for **DL-Lite<sub>A</sub>**.

This approach has several drawbacks:

- The technique is no more AC<sub>0</sub> in the data complexity, since the ABox  $A_{\mathbf{M}, \mathbf{D}}$  to materialize is in general **polynomial** in the size of the data.
- $A_{\mathbf{M}, \mathbf{D}}$  may be very large, and thus it may be infeasible to actually materialize it.
- **Freshness** of  $A_{\mathbf{M}, \mathbf{D}}$  with respect to the underlying data source(s) may be an issue.

# Top-down approach to query answering

Given an OBDM instance  $\mathbf{O} = \langle \mathbf{P}, \mathbf{D} \rangle$  with  $\mathbf{P} = \langle \mathbf{T}, \mathbf{M}, \mathbf{S} \rangle$  and a UCQ  $q$ , this approach consists of three steps:

1. **Reformulation:** Compute the **perfect rewriting**  $q_{pr} = \text{PerfectRef}(q, \mathbf{T})$  of the query  $q$ , using the assertions of the TBox  $\mathbf{T}$ . As we know, the perfect rewriting  $q_{pr}$  is such that  $\text{cert}(q, \langle \mathbf{T}, \mathbf{A} \rangle) = \text{Eval}_{\text{cwa}}(q_{pr}, \mathbf{A})$  for each ABox  $\mathbf{A}$ .
2. **Unfolding:** Compute from  $q_{pr}$  a new query  $q_{unf}$  by **unfolding**  $q_{pr}$  using (the split version of) the mappings  $\mathbf{M}$  (as for GAV information integration systems)
  - Essentially we are going to replace each atom in  $q_{pr}$  with the corresponding query  $\Phi$  over the database  $\mathbf{D}$  in  $\mathbf{M}$  - in fact  $\mathbf{M}$  is a GAV mapping so the predicates of  $\mathbf{T}$  (and hence of  $q_{pr}$ ) are defined in  $\mathbf{M}$  as views.
  - The query  $q_{unf}$  is such that  $\text{Eval}(q_{unf}, \mathbf{D}) = \text{Eval}_{\text{cwa}}(q_{pr}, \mathbf{A}_{\mathbf{M}, \mathbf{D}})$  for each database  $\mathbf{D}$ .
3. **Evaluation:** Delegate the evaluation of  $q_{unf}$  to the relational **DBMS** managing  $\mathbf{D}$ .

# Computational complexity of query answering

From the top-down approach to query answering, and the complexity results for DL-Lite, we obtain the following result

**Theorem:** In a DL-Lite OBDM instance  $\mathbf{O} = \langle \mathbf{P}, \mathbf{D} \rangle$  with  $\mathbf{P} = \langle \mathbf{T}, \mathbf{M}, \mathbf{S} \rangle$  query answering (i.e., the recognition problem) of a UCQ  $\mathbf{q}$  is:

- **NP-complete** in the size of the query  $\mathbf{q}$ .
- **PTime** in the size of the **TBox**  $\mathbf{T}$  and the **mappings**  $\mathbf{M}$ .
- **AC<sub>0</sub>** in the size of the database  $\mathbf{D}$ .

**Note:** The AC<sub>0</sub> result is a consequence of the fact that query answering in such a setting can be reduced to evaluating an SQL query over the relational database  $\mathbf{D}$ .

## Credits

The presentations of this module of the course are based on the original slides of Prof. Giuseppe De Giacomo, Prof. Diego Calvanese, Prof. Marco Console, Prof. Maurizio Lenzerini, and Prof. Domenico Lembo.