SEMINARIO INGEGNERIA DEL SOFTWARE
a.a. 2006/2007

INTERNET MOVIE DATABASE

Studente: Cristiano Sticca
Docente: Prof. Giuseppe De Giacomo

OBIETTIVI

- Creazione di un DB locale rappresentante tutti i film prodotti dall'inizio della cinematografia ad oggi
- Generazione schema ER dell' "INTERNET MOVIE DB"
- Realizzazione schema logico
- Realizzazione schema fisico
- Creazione ontologia in DL-Lite$_A$
- Creazione mapping tra DB e ontologia
- Verifica su QuOnto

PROBLEMATICHE

- Realizzazione di parser java per la creazione di script SQL
- Identificazione struttura dei file importati dal sito http://imdb.com/interfaces#win32
- Tempo per popolare il DB!!!

File from IMDb…

Il sito www.imdb.com permette il download di determinati file che rappresentano il contenuto del DB su internet al seguente indirizzo http://imdb.com/interfaces

Come accennato precedentemente, il problema sostanziale è che tali file (40 circa) hanno una formattazione pressochè diversa l'uno dall'altro.

E questo quindi ha comportato la realizzazione di altrettanti parser scritti in Java.

File from IMDb… an example

Vediamo come possono essere strutturati alcuni di questi file.

Consideriamo i seguenti: movies.LIST and actors.LIST

Per una questione di leggibilità sono stati rinominati in .txt

movies.LIST

CRC: 0x9A686F55 File: movies.list Date: Fri Sep 14 01:00:00 2007

http://www.imdb.com
movies.list
2007-09-12

MOVIES LIST
==========

#1 (2005)		2005
#1 Fan: A Darkomentary (2005) (V)	2005	
#28 (2002)		2002
#2: Drops (2004)	2004	
#7 Train: An Immigrant Journey, The (2000)	2000	
$ (1971)	1971	
$1,000 Reward (1913)		1913
$1,000 Reward (1915)		1915
$1,000 Reward (1923)		1923
$1,000,000 Reward, The (1920)		1920
$10,000 Under a Pillow (1921)		1921

Parser Java per movies.LIST

Crea lo Script SQL

Legge il file sorgente

Inserisce valori nello script

```
public class movies {
        public static void main (String[] args) throws Exception{
                String primo,secondo;
                secondo = null;
                int movieid=0;
                String[]res;

                 FileWriter fi = new FileWriter("G:\\Documents and Settings\\cristiano\\Desktop\\tesina seminario\\tesina\\finalScripts\\movies.sql");
                PrintWriter out=new PrintWriter(fi);
                FileReader f = new FileReader("G:\\Documents and Settings\\cristiano\\Desktop\\tesina seminario\\tesina\\file IMDB\\movies.txt");
                BufferedReader filebuf = new BufferedReader(f);

                String nextStr;
                nextStr = filebuf.readLine();

                  while (!nextStr.equals("MOVIES LIST")){
                nextStr = filebuf.readLine();
                }
                nextStr = filebuf.readLine();
                nextStr = filebuf.readLine();
                 out.print("INSERT IGNORE INTO movies VALUES\n");
                 while (nextStr!=null){
                                        if (nextStr.equals("-------------------------------------------------------------------------------")){
                                                System.out.println("Movies inseriti nel db");
                                                break;
                                        }
```

Utilizziamo i “multiple inserts” per velocizzare le operazioni in fase di popolamento

```
        nextStr=nextStr.replace("'","");

        res=nextStr.split("\t");

          for (int x=1; x<res.length; x++){

          if (!res[x].equals("\t")){
                        secondo= res[x];
          }
          }

          primo=res[0];
          if (primo.contains("(")){
                        int indice = primo.indexOf("(");
                        primo = primo.substring(0,indice);
           }
           out.append("('"+movieid+"','"+primo+"','"+secondo+"',null,null,null,null,null,null,null,null,null),\n");

          nextStr = filebuf.readLine();// legge una riga del file
          movieid++;
           }

        filebuf.close();  // chiude il file
        out.close();
    }
}
```

actors.LIST (relativo alle attrici)

THE ACTRESSES LIST
==========

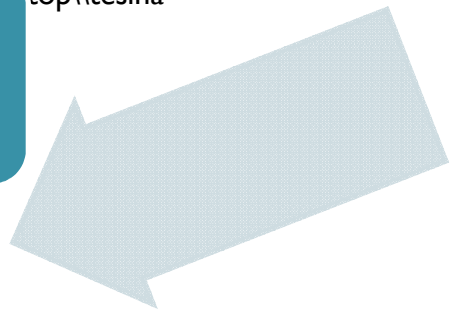
Name	Titles
----	------
'La Mueque'	Tarantos, Los (1963) [Cantaor] <17>
'La Tata' Castro, Maria Tereza	"Granja tolima, La" (2004) [Herself]
'La Veneno', Cristina	Secreto de la veneno, El (1997) (V) <1> Venganza de la veneno, La (1997) (V) <1> "Aquí hay tomate" (2003) {(2006-10-11)} [Herself]
't Hart, Josine	Barbiere di Siviglia, Il (1992) (TV) [Mime artists] <11> Nieuwe moeder, De (1996) [Vrouw in trein] <23>
't Seyen, Hilda	Moedige bruidegom, De (1952) [Moeder van Tinneke]

Parser Java per actors.LIST

```
public class actresses {
        public static void main (String[] args) throws Exception{
                                        /*variabili*/
         FileWriter fi = new FileWriter("G:\\Documents and Settings\\cristiano\\Desktop\\tesina
        seminario\\tesina\\finalScripts\\actresses.sql");
        PrintWriter out=new PrintWriter(fi);
        FileWriter fi3 = new FileWriter("G:\\Documents and Settings\\cristiano\\Desktop\\tesina
        seminario\\tesina\\finalScripts\\actresses2movie.sql");
        PrintWriter out3=new PrintWriter(fi3);

        FileReader f = new FileReader("G:\\Documents and Se                    top\\tesina
        seminario\\tesina\\file IMDB\\actresses.txt");
        BufferedReader filebuf = new BufferedReader(f);
          Connection conn2 = null;
        ResultSet rs3=null;
 try
          {
String userName = "root";
String password = "041524";
String url = "jdbc:mysql://localhost/imdb";
Class.forName ("com.mysql.jdbc.Driver").newInstance ();
conn2 = DriverManager.getConnection (url, userName, password);
//System.out.println ("Database connection established");
          }
      catch (Exception e)
      {
      e.printStackTrace();
System.err.println ("Cannot connect to database server");
      }
```

Connessione al DB



```
out.print("INSERT IGNORE INTO actor VALUES\n");
out3.print("INSERT IGNORE INTO actor2movie VALUES\n");
                    String nextStr;
      nextStr = filebuf.readLine();
                                 while (!nextStr.equals("THE ACTRESSES LIST")) {
                                              nextStr=filebuf.readLine();
                                 }
                                 nextStr = filebuf.readLine();
                                  nextStr = filebuf.readLine();
                                        nextStr = filebuf.readLine();
                                        nextStr = filebuf.readLine();
                                        nextStr = filebuf.readLine();
                                 }
                 while (nextStr!=null){
                                                          try{
                                   if (nextStr.equals("SUBMITTING UPDATES")){
                                              System.out.println("Actresses inseriti nel db");
                                              break;
                                 }
                                              nextStr = nextStr.replace("'","");
                                              if(nextStr.equals("")){
                                                          attoreid++;
                                                          nextStr=filebuf.readLine();
                                              }
                                              if (nextStr.startsWith("\t")){
                                                          secondo=nextStr;
                                                          secondo=secondo.replace("'","");
                                                          secondo=secondo.replace("\t","");
                                                          if (secondo.contains("(")){
                                                                      int aperta=secondo.indexOf("(");
                                                                      int chiusa=secondo.indexOf(")");
                                                                      year = secondo.substring(aperta+1,chiusa);
                                                           }
```

Utilizziamo i "multiple inserts" per velocizzare le operazioni in fase di popolamento

```
else {year=null; }
                if (secondo.contains("[")){
                        int aperta2=secondo.indexOf("[");
                        int chiusa2=secondo.indexOf("]");
                        as = secondo.substring(aperta2+1,chiusa2);    }
                 else {as=null; }}
else {
                primo=nextStr;
                primo=primo.replace("'","");
                int tab=primo.indexOf("\t");
                nomecognome = primo.substring(0,tab);
                if (nomecognome.contains(",")){
                        int virgola=nomecognome.indexOf(",");
                        cognome = nomecognome.substring(0,virgola);
                        nome= nomecognome.substring(virgola+2,tab); }
                else {nome = null;
                        cognome = nomecognome; }
                out.append("('"+attoreid+"','"+nome+"','"+cognome+"','F'),\n");
                secondo=primo.substring(tab);
                secondo=secondo.replace("'","");
                secondo=secondo.replace("\t","");
                if (secondo.contains("(")){
                        int aperta=secondo.indexOf("(");
                        int chiusa=secondo.indexOf(")");
                        year = secondo.substring(aperta+1,chiusa); }
                 else {year=null;}
                if (secondo.contains("[")){
                        int aperta2=secondo.indexOf("[");
                        int chiusa2=secondo.indexOf("]");
                        as = secondo.substring(aperta2+1,chiusa2);
                 }
                 else {
                        as=null;
                 }
```

```
}
if (secondo.contains("[")){
                int quadra=secondo.indexOf("[");
                secondo=secondo.substring(0,quadra);
                 }
if (secondo.contains("(")){
                int tonda=secondo.indexOf("(");
                secondo=secondo.substring(0,tonda);
                 }
                        try{
                                Statement s2 = conn2.createStatement();

                rs3=s2.executeQuery("SELECT id FROM movies WHERE title='"+secondo+"' and year='"+year+"'");

                                rs3.last();
                        movieid =(Integer)rs3.getObject(1);
                                }
                                catch(Exception e){

                                }
          out3.append("('"+attoreid+"','"+movieid+"','"+as+"'),\n");
   nextStr = filebuf.readLine();// legge una riga del file
}
catch(Exception e){
                e.printStackTrace();
                break;
}
    }
   filebuf.close();  // chiude il file
   out.close();
   }
                }
```

query al DB per trovare l'id del film

"multiple insert" nella relazione actor2movie

script SQL (movies)

NOTARE L'INSERIMENTO MULTIPLO

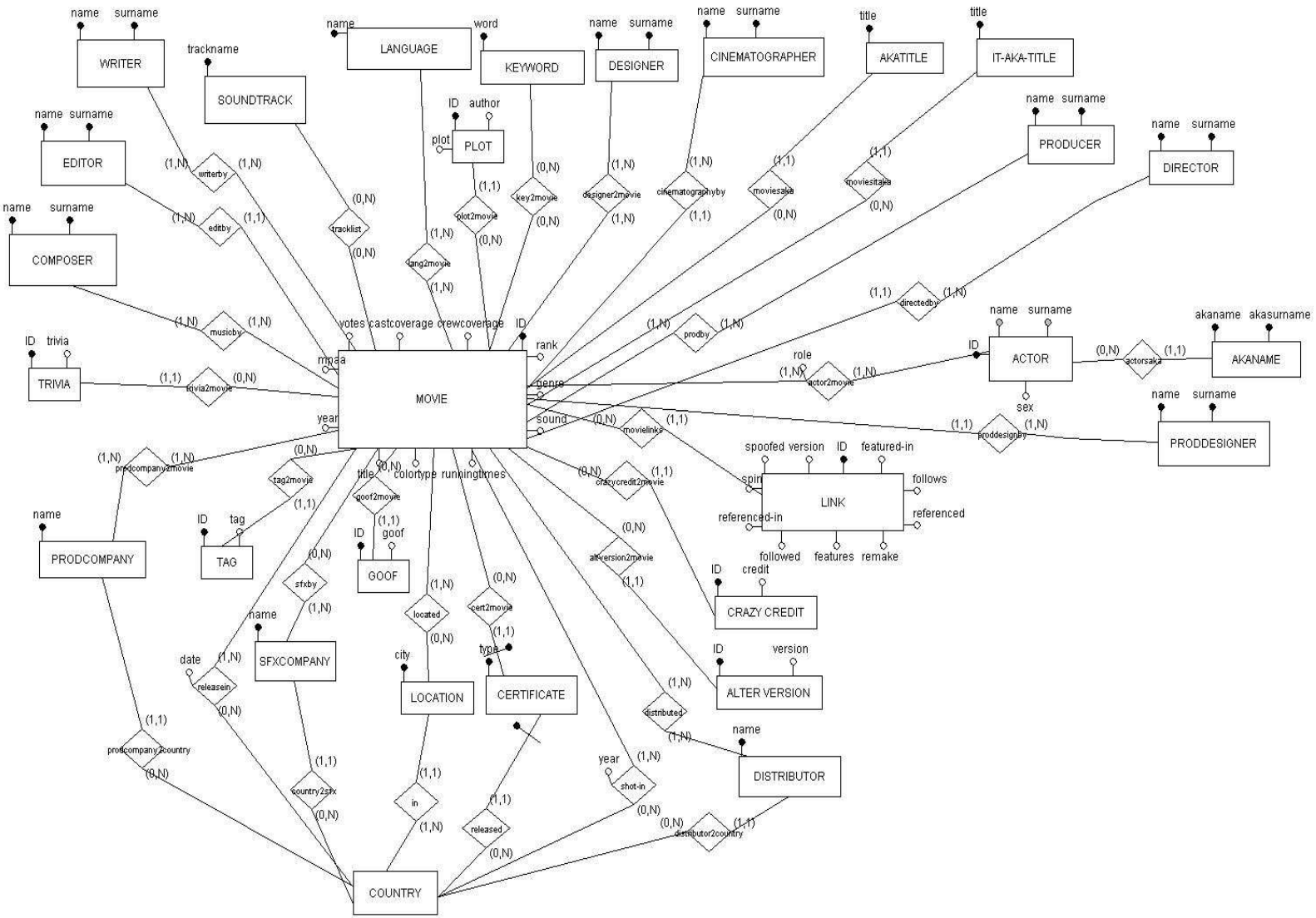
```
INSERT IGNORE INTO movies VALUES
('1','#1 ','2005',null,null,null,null,null,null,null,null,null),
('2','#1 Fan: A Darkomentary ','2005',null,null,null,null,null,null,null,null,null),
('3','#28 ','2002',null,null,null,null,null,null,null,null,null),
('4','#2: Drops ','2004',null,null,null,null,null,null,null,null,null),
('5','#7 Train: An Immigrant Journey, The
','2000',null,null,null,null,null,null,null,null,null),
('6','#Bfl O {ggGX = STwWcfl x 2s4 ','1963',null,null,null,null,null,null,null,null,null),
('7','$ ','1971',null,null,null,null,null,null,null,null,null),
('8','$1,000 Reward ','1913',null,null,null,null,null,null,null,null,null),
('9','$1,000 Reward ','1915',null,null,null,null,null,null,null,null,null),
('10','$1,000 Reward ','1923',null,null,null,null,null,null,null,null,null),
('11','$1,000,000 Reward, The ','1920',null,null,null,null,null,null,null,null,null),
('12','$10,000 Under a Pillow ','1921',null,null,null,null,null,null,null,null,null),
('13','$100 & a T-Shirt: A Documentary About Zines in the Northwest
','2004',null,null,null,null,null,null,null,null,null),
('14','$100,000 ','1915',null,null,null,null,null,null,null,null,null),
('15','$100,000 Bill, The ','1915',null,null,null,null,null,null,null,null,null),
('17','$100,000 Pyramid, The ','2001',null,null,null,null,null,null,null,null,null),
```

script SQL (actors)

INSERT IGNORE INTO actor VALUES('1','null','La Mueque','F'),('2','Maria Tereza','La Tata Castro','F'),('3','Cristina','La Veneno','F'),('4','Josine','t Hart','F'),('5','Hilda','t Seyen','F'),('6','Moni','ya','F'),('7','Martha','103','F'),('8','Sharon','10X','F'),('9','Nicole','11:11','F'),('10','null','12 Elite Girls','F'),('11','Die','12 Elite Girls','F'),('12','Rachel','18','F'),('13','null','1988 Montclair Squad','F'),('14','Die','20 Wienerinnen','F'),('15','null','2004 Buffalo Jills Cheerleaders','F'),('16','Diamond','4 Ever','F'),('17','null','4 Non Blondes','F'),('18','Colt','45','F'),('19','Cat','9','F'),('20','Kat','9','F'),('21','Suilma','AAlitaleb','F'),('22','Mrs.','ACosta','F'),('23','Michele','ACourt','F'),

INSERT IGNORE INTO actor2movie VALUES
('91784','425331','Edith'),('91784','443564','Kamen'),('91784','478275','Sister Candida'),('91784','478583','null'),('91785','33484','Corinna Gerber'),('91785','178468','null'),('91785','278096','null'),('91785','314761','Rosi'),('91785','314810','null'),('91785','383550','null'),('91785','497686','Erika Hanson'),('91785','536953','Selma Kremer'),

Schema ER


WRITER
SOUNDTRACK
LANGUAGE
KEYWORD
DESIGNER
CINEMATOGRAPHER
AKATITLE
IT-AKA-TITLE
EDITOR
PLOT
PRODUCER
DIRECTOR
COMPOSER
MOVIE
ACTOR
AKANAME
TRIVIA
PRODDESIGNER
LINK
PRODCOMPANY
TAG
GOOF
CRAZY CREDIT
SFXCOMPANY
LOCATION
CERTIFICATE
ALTER VERSION
DISTRIBUTOR
COUNTRY
writerby
trackname
language
keyword
designer
cinematographer
akatitle
it-aka-title
editby
plot
producer
director
composerby
tracklist
key2movie
designer2movie
cinematographyby
moviesaka
moviesitaka
plot2movie
lang2movie
votes
castcoverage
crewcoverage
rank
rmaa
genre
sound
year
musicby
directedby
prodby
role
actor2movie
actorsaka
akaname
akasurname
sex
trivia2movie
movielinks
proddesignby
prodcompany2movie
tag2movie
goof2movie
title
colortype
runningtimes
crazycredit2movie
spoofed version
featured-in
spin
follows
referenced-in
referenced
followed
features
remake
altversion2movie
credit
sfxby
located
cert2movie
releasein
city
type
distributed
version
prodcompany2country
country2sfx
in
released
shot-in
distributor2country

Schema Logico

MOVIES(ID,title,year,runtime,sound,genre,colortype,castcoverage,crewcoverage,votes,rank,mpaa)

inclusione: MOVIES[ID] ⊆ releasein[MOVIE]
inclusione: MOVIES[ID] ⊆ musicby[MOVIE]
inclusione: MOVIES[ID] ⊆ lang2movie[MOVIE]
inclusione: MOVIES[ID] ⊆ writerby[MOVIE]
inclusione: MOVIES[ID] ⊆ prodcompany2movie[MOVIE]
inclusione: MOVIES[ID] ⊆ actor2movie[MOVIE]
inclusione: MOVIES[ID] ⊆ prodby[MOVIE]
inclusione: MOVIES[ID] ⊆ designer2movie[MOVIE]
inclusione: MOVIES[ID] ⊆ distributor[MOVIE]
inclusione: MOVIES[ID] ⊆ located[MOVIE]
inclusione: MOVIES[ID] ⊆ shotin[MOVIE]
FK: MOVIES[ID] ⊆ editby[MOVIE]
FK: MOVIES[ID] ⊆ proddesignby[MOVIE]
FK: MOVIES[ID] ⊆ directedby[MOVIE]
FK: MOVIES[ID] ⊆ cinematographyby[MOVIE]

……

Schema Fisico

```
create table movies (
        id int(8) primary key,
        title varchar(200),
        year varchar(100),
        runtimes varchar(100),
        sound varchar(100),
        genre varchar(100),
        colortype varchar(100),
        crewcoverage varchar(100),
        castcoverage varchar(100),
        votes int(8),
        rank float,
        mpaa longtext,
        index(title),
        index(title,year),
        unique(title,year),
        check (id in (select movie from musicby)),
        check (id in (select movie from writerby)),
        check (id in (select movie from actor2movie)),
        check (id in (select movie from prodby)),
        check (id in (select movie from designer2movie)),
        check (id in (select movie from distributor)),
        check (id in (select movie from located)),
        check (id in (select movie from shotin))
        check (id in (select movie from releasein))
        );
```

Ontologia

- "*Descrizione formale esplicita dei concetti di un dominio*".
- Essa contiene:
  - Un insieme di classi (concetti rilevanti)
  - Un insieme di relazioni tra queste classi
  - Un insieme di proprietà attribuite a ciascun concetto
  - Un insieme di restrizioni sulle proprietà

OWL (ontology web language)

OWL Ontology Web Language, è una raccomandazione del W3C.

- Linguaggio per esprimere le ontologie

- Tre tipi di OWL:
  - OWL Lite, versione sintatticamente più semplice che permette di esprimere una gerarchia di classi e semplici restrizioni
  - OWL DL, versione intermedia basata sulle logiche descrittive, offre un potere espressivo elevato mantenendo completezza e decidibilità
  - OWL FULL, offre la massima espressività senza offrire alcuna garanzia circa completezza e decidibilità

Description Logic

Le logiche descrittive (DL) sono frammenti decidibili della FOL per esprimere la conoscenza in termini di:

- concetti atomici (predicati unari)
- ruoli atomici (predicati binari)
- individui (costanti)

Una base di conoscenza in DL comprende:

- *TBox*: insieme di assiomi terminologici, ovvero il vocabolario del dominio applicativo (concetti e ruoli)
- *ABox*: contiene asserzioni circa gli individui che popolano il mondo in oggetto, assegnando loro un nome e asserendo le loro proprietà

DL-Lite family

• Description Logics (DLs) underlie the standard ontology languages for the Semantic Web (i.e., OWL, OWL-DL)

• *DL-Lite is a family of DLs optimized according to the tradeoff between expressive* power and data complexity

• Two maximal languages that enjoy FOL-rewritability: *DL-Lite*$_F$, *DL-Lite*$_R$
(we use simply *DL-Lite to refer to both languages)*

• With minimal additions to *DL-Lite, data complexity jumps to NLOGSPACE or* above
→We lose FOL-rewritability

Provides an answer to our basic question: *For which ontology languages can we answer queries over an ontology efficiently (in data complexity)?*

DL-Lite$_F$

DL-LiteF

Ontology language:

• Concept inclusion assertions: Cl $\sqsubseteq$ Cr, with:

Cl $\rightarrow$ A | $\exists$R | Cl 1 $\sqcap$ Cl 2

Cr $\rightarrow$ A | $\exists$R | $\neg$A | $\neg\exists$R

R $\rightarrow$ P | P$^-$

• Functionality assertions: (funct R)

Database facts: A(c), P(c, d), with c, d constants

Observations:

• Captures all the basic constructs of ER and UML Class Diagrams

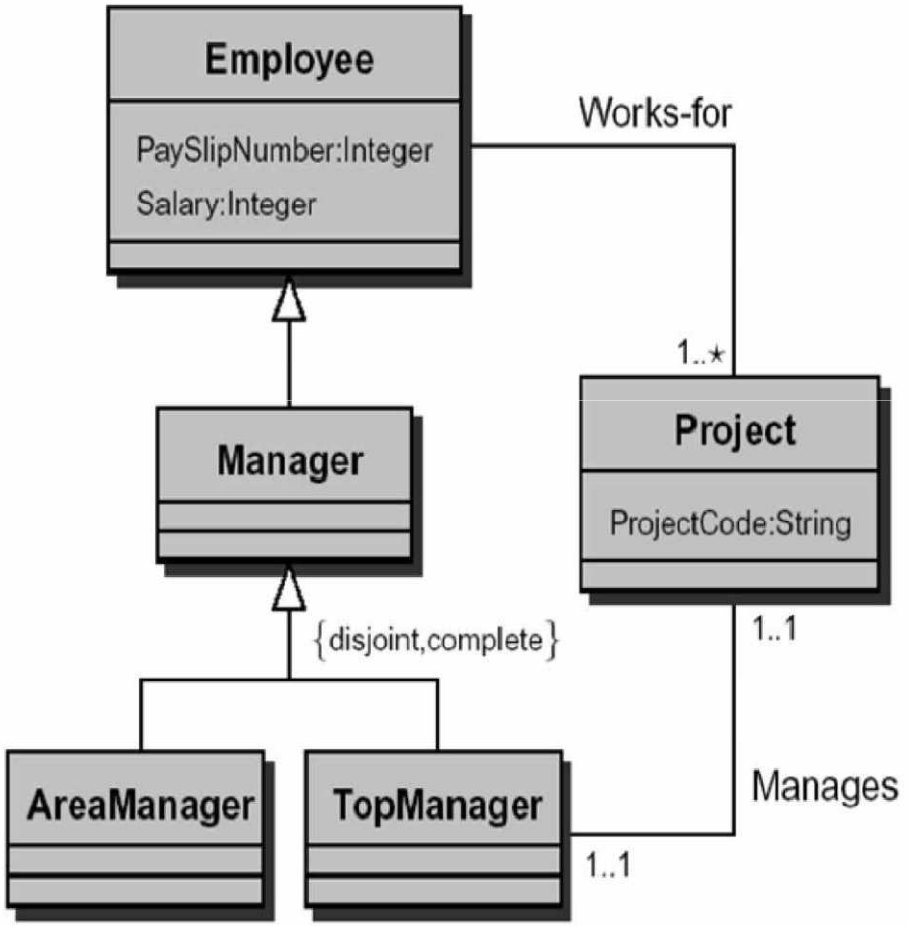
• Notable exception: covering constraints in generalizations

…continua DL-Lite

- **Capturing basic ontology constructs in *DL-Lite***
- ISA between classes → $A1 \sqsubseteq A2$
- disjointness between classes → $A1 \sqsubseteq \neg A2$
- domain and range of relations → $\exists P \sqsubseteq A1 \quad \exists P^- \sqsubseteq A2$
- mandatory participation → $A1 \sqsubseteq \exists P \; A2 \sqsubseteq \exists P^-$
- functionality of relations (in *DL-Lite*$_F$) → *(funct P) (funct P)*
- ISA between relations (in *DL-Lite*$_R$) → $R1 \sqsubseteq R2$

DL-Lite example

Manager ⊑ Employee
AreaManager ⊑ Manager
TopManager ⊑ Manager
AreaManager ⊑ ¬TopManager
∃WorksFor ⊑ Employee
∃WorksFor⁻ ⊑ Project
Project ⊑ ∃WorksFor⁻
(funct WorksFor)
(funct WorksFor⁻)
...
Note: in *DL-Lite we cannot capture completeness of the hierarchy*



DL-Lite$_R$

Ontology language:

• Concept inclusion assertions: $Cl \sqsubseteq Cr$, with:

$Cl \rightarrow A \mid \exists R \mid Cl\ 1 \sqcap Cl\ 2$

$Cr \rightarrow A \mid \exists R \mid \neg A \mid \neg \exists R \mid \exists R.\mathbf{A}$

$R \rightarrow P \mid P^-$

• Role inclusion assertions: $R1 \sqsubseteq R2$

Database facts: A(c), P(c, d), with c, d constants

Properties:

• Drops functional restrictions in favor of ISA between roles

• Extends (the DL fragment of) RDFS

Query answering in DL-Lite

Given a CQ q, an ontology O, and a database D, we compute cert(q,O,D) as follows:

1. Close ontology O wrt disjointness assertions and check for satisfiability wrt D
2. Using O, reformulate CQ q as a union $r_{q,O}$ of CQs
3. Evaluate $r_{q,O}$ directly over D using the RDBMS

Correctness of this algorithm shows FOL-rewritability of query answering in *DL-Lite*

→Query answering over *DL-Lite ontologies can be done using RDBMS technology*

→Prototype system implemented: QuOnto

DL-Lite complexity results

- Consistency checking is
  - **polynomial** in the size of the **ontology** and of the **database**

- Query answering is
  - **exponential** in the size of the **query** (NP-complete)
  - **polynomial** in the size of the **ontology** and of the **database** (in fact LOGSPACE in the database)

Can we further extend these results to more expressive ontology languages?

Essentially NO!

DL-Lite$_A$

To speak about DL-Lite$_A$ we first have to introduce the DL *DL-Lite$_{FR}$, that combines the main features of two DLs*

presented previously, called *DL-Lite$_F$ and DL-Lite$_R$ respectively, and forms the basics of DL-Lite$_A$. In providing the specification of our logics, we use the following notation:*

– A denotes an atomic concept, B a basic concept, and C a general concept;

– D denotes an atomic value-domain, E a basic value-domain, and F a general value-domain;

– P denotes an atomic role, Q a basic role, and R a general role;

– U_C denotes an atomic concept attribute, and V_C a general concept attribute;

– U_R denotes an atomic role attribute, and V_R a general role attribute;

– T_C denotes the universal concept, T_D denotes the universal value-domain.

Given a concept attribute *U_C (resp. a role attribute U_R), we call the domain of U_C* (resp. *U_R), denoted by $\delta(U_C)$ (resp. $\delta(U_R)$), the set of objects (resp. of pairs of objects)* that *U_C (resp. U_R) relates to values, and we call range of U_C (resp. U_R), denoted* by *$\rho(U_C)$ (resp. $\rho(U_R)$), the set of values that U_C(resp. U_R) relates to objects (resp.* pairs of objects). Notice that the domain *$\delta(U_C)$ of a concept attribute U_C is a concept,* whereas the domain *$\delta(U_R)$ of a role attribute U_R is a role. Furthermore, we denote* with *$\delta_F(U_C)$ (resp. $\delta_F(U_R)$) the set of objects (resp. of pairs of objects) that U_C (resp. U_R) relates to values in the value-domain F. In particular, DL-Lite$_{FR}$ expressions are* defined as in the next slide.

• • •

– **Concept expressions:**

$B ::= A \mid \exists Q \mid \delta(U_C)$

$C ::= T_C \mid B \mid \neg B \mid \exists Q.C \mid \delta_F(U_C) \mid \exists \delta_F(U_R) \mid \exists \delta_F(U_R)^-$

– **Value-domain expressions** (*rdfDataType denotes predefined value-domains such*

as integers, strings, etc.):

$E ::= D \mid \rho(U_C) \mid \rho(U_R)$

$F ::= T_D \mid E \mid \neg E \mid rdfDataType$

– **Attribute expressions**:

$VC ::= U_C \mid \neg U_C$

$VR ::= U_R \mid \neg U_R$

– **Role expressions:**

$Q ::= P \mid P^- \mid \delta(U_R) \mid \delta(U_R)^-$

$R ::= Q \mid \neg Q \mid \delta_F(U_R) \mid \delta_F(U_R)^-$

• • •

A *DL-Lite$_{FR}$ knowledge base (KB) K = <T ,A> is constituted by two components:*

a TBox *T , used to represent intensional knowledge, and an ABox A, used to represent* extensional knowledge. *DL-Lite$_{FR}$ TBox assertions are of the form:*

$B \subseteq C$ concept inclusion assertion

$Q \subseteq R$ role inclusion assertion

$E \subseteq F$ value-domain inclusion assertion

$U_C \subseteq V_C$ concept attribute inclusion assertion

$U_R \subseteq V_R$ role attribute inclusion assertion

(funct *P) role functionality assertion*

(funct *P^-) inverse role functionality assertion*

(funct *U_C) concept attribute functionality assertion*

(funct *U_R) role attribute functionality assertion*

• • •

A DL-Lite$_A$ knowledge base is pair <T , A>, where A is a DL-Lite$_{FR}$ ABox, and T is a DL-Lite$_{FR}$ TBox satisfying the following conditions:

1. for every atomic or inverse of an atomic role Q appearing in a concept of the form ∃ Q.C, the assertions (funct Q) and (funct Q$^-$) are not in T ;

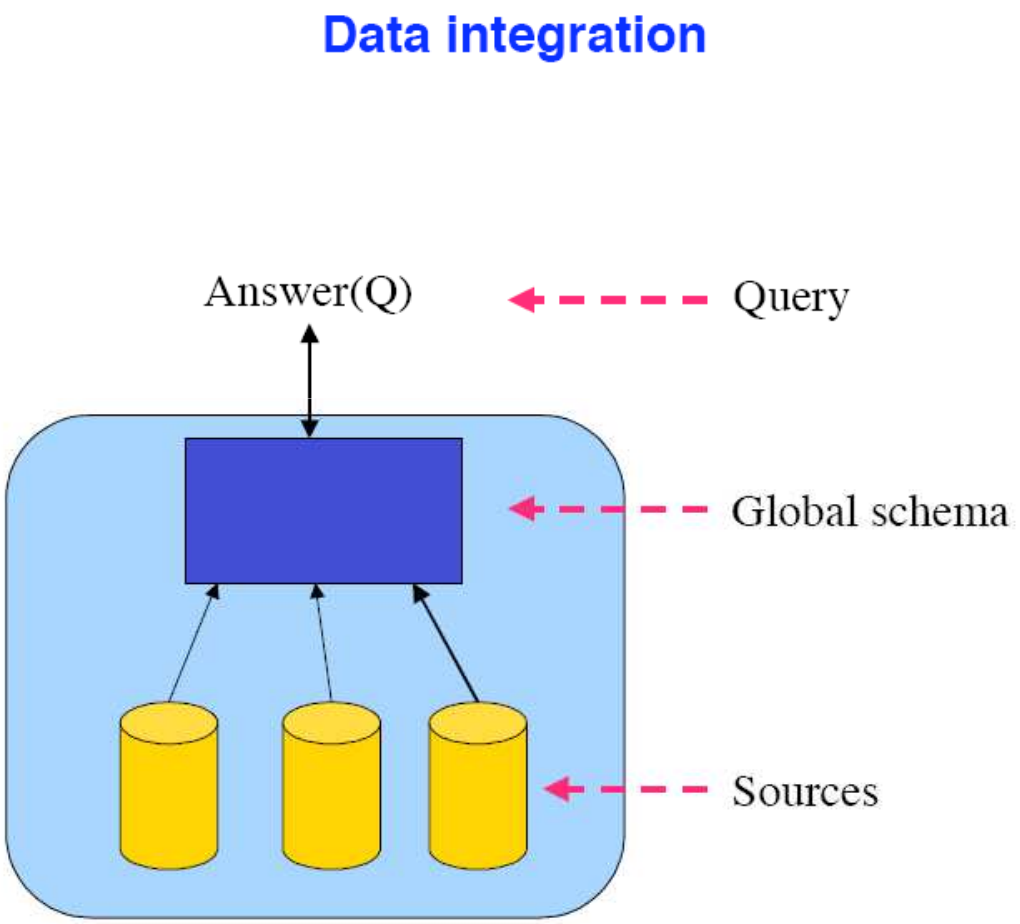
2. for every role inclusion assertion Q ⊆ R in T , where R is an atomic role or the inverse of an atomic role, the assertions (funct R) and (funct R$^-$) are not in T ;

3. for every concept attribute inclusion assertion $U_C \subseteq V_C$ in T , where V_C is an atomic concept attribute, the assertion (funct V_C) is not in T ;

4. for every role attribute inclusion assertion $U_R \subseteq V_R$ in T , where V_R is an atomic role attribute, the assertion (funct V_R) is not in T .

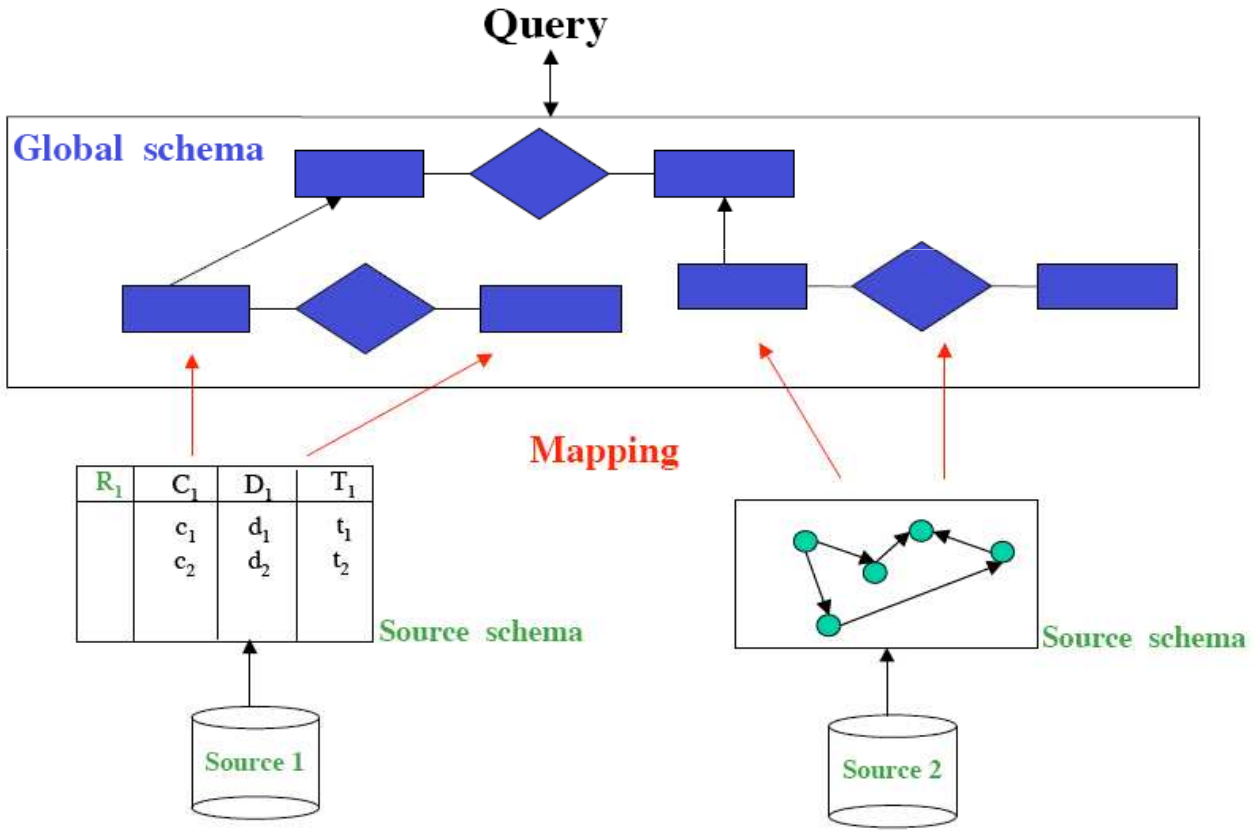
Roughly speaking, a *DL-Lite$_A$ TBox imposes the condition that every functional role cannot be specialized by using it in the right-hand side of role inclusion assertions; the* same condition is also imposed on every functional (role or concept) attribute. It can be shown that functionalities specified in a *DL-Lite$_A$ TBox are not implicitly propagated* in the TBox, and that this allows for LOGSPACE query answering.

Mapping… what is this?

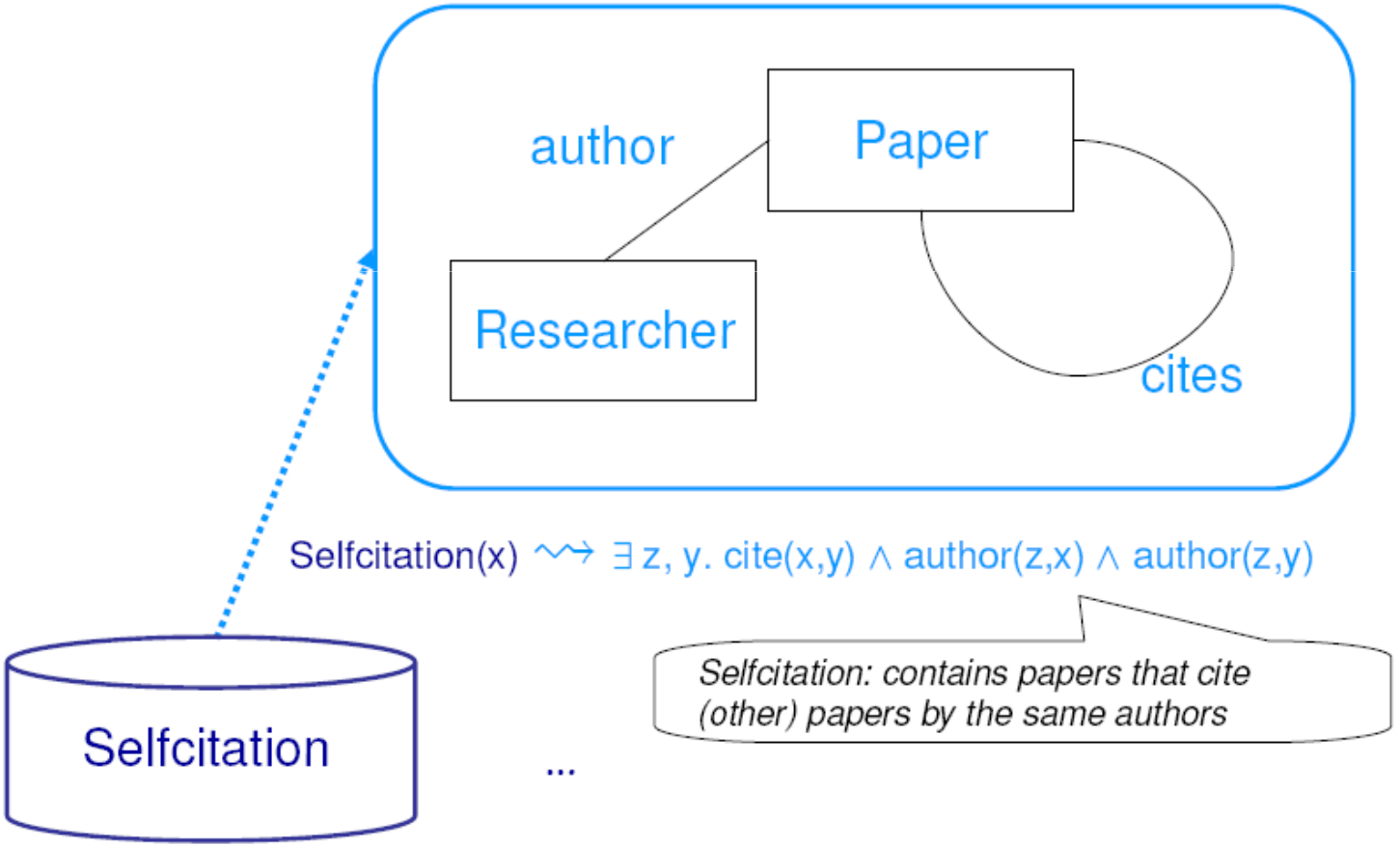


• • •

Data integration



An example


author
Paper
Researcher
cites
Selfcitation(x) ⇝ ∃ z, y. cite(x,y) ∧ author(z,x) ∧ author(z,y)
Selfcitation
Selfcitation: contains papers that cite (other) papers by the same authors
...

Mapping…how many?

How is the mapping $\mathcal{M}$ between $\mathcal{S}$ and $\mathcal{G}$ specified?

- Are the sources defined in terms of the global schema?

  Approach called **source-centric**, or **local-as-view**, or **LAV**

- Is the global schema defined in terms of the sources?

  Approach called **global-schema-centric**, or **global-as-view**, or **GAV**

- A mixed approach?

  Approach called **GLAV**

GAV vs LAV example

Global schema: movie$(Title, Year, Director)$

european$(Director)$

review$(Title, Critique)$

Source 1: $r_1(Title, Year, Director)$ since 1960, European directors

Source 2: $r_2(Title, Critique)$ since 1990

Query: Title and critique of movies in 1998

$\exists D.\ \text{movie}(T, 1998, D) \wedge \text{review}(T, R)$, written

$\{\ (T, R) \mid \text{movie}(T, 1998, D) \wedge \text{review}(T, R)\ \}$

LAV formalization

In LAV (with sound sources), the mapping $\mathcal{M}$ is constituted by a set of assertions:

$$s \rightsquigarrow \phi_{\mathcal{G}}$$

one for each source element s in $\mathcal{A}_{\mathcal{S}}$, where $\phi_{\mathcal{G}}$ is a **query** over $\mathcal{G}$ of the arity of s.

Given source database $\mathcal{C}$, a database $\mathcal{B}$ for $\mathcal{G}$ satisfies $\mathcal{M}$ wrt $\mathcal{C}$ if for each $s \in \mathcal{S}$:

$$s^{\mathcal{C}} \subseteq \phi_{\mathcal{G}}{}^{\mathcal{B}}$$

In other words, the assertion means $\forall \vec{\mathbf{x}} \left(s(\vec{\mathbf{x}}) \rightarrow \phi_{\mathcal{G}}(\vec{\mathbf{x}}) \right)$.

The mapping $\mathcal{M}$ and the source database $\mathcal{C}$ do **not** provide direct information about which data satisfy the global schema. Sources are views, and we have to answer queries on the basis of the available data in the views.

LAV example

Global schema: movie$(Title, Year, Director)$
european$(Director)$
review$(Title, Critique)$

LAV: associated to source relations we have views over the global schema

$$\mathsf{r}_1(T, Y, D) \rightsquigarrow \{ (T, Y, D) \mid \text{movie}(T, Y, D) \wedge \text{european}(D) \wedge Y \geq 1960 \}$$

$$\mathsf{r}_2(T, R) \rightsquigarrow \{ (T, R) \mid \text{movie}(T, Y, D) \wedge \text{review}(T, R) \wedge Y \geq 1990 \}$$

The query $\{ (T, R) \mid \text{movie}(T, 1998, D) \wedge \text{review}(T, R) \}$ is processed by means of an inference mechanism that aims at re-expressing the atoms of the global schema in terms of atoms at the sources. In this case:

$$\{ (T, R) \mid \mathsf{r}_2(T, R) \wedge \mathsf{r}_1(T, 1998, D) \}$$

Gav formalization

In GAV (with sound sources), the mapping $\mathcal{M}$ is constituted by a set of assertions:

$$g \rightsquigarrow \phi_{\mathcal{S}}$$

one for each element g in $\mathcal{A}_{\mathcal{G}}$, where $\phi_{\mathcal{S}}$ is a **query** over $\mathcal{S}$ of the arity of g.

Given source database $\mathcal{C}$, a database $\mathcal{B}$ for $\mathcal{G}$ satisfies $\mathcal{M}$ wrt $\mathcal{C}$ if for each $g \in \mathcal{G}$:

$$g^{\mathcal{B}} \supseteq {\phi_{\mathcal{S}}}^{\mathcal{C}}$$

In other words, the assertion means $\forall \vec{\mathbf{x}} \, (\phi_{\mathcal{S}}(\vec{\mathbf{x}}) \rightarrow g(\vec{\mathbf{x}}))$.

Given a source database, $\mathcal{M}$ provides direct information about which data satisfy the elements of the global schema. Relations in $\mathcal{G}$ are views, and queries are expressed over the views. Thus, it seems that we can simply evaluate the query over the data satisfying the global relations (as if we had a single database at hand).

Gav example

Global schema: movie$(Title, Year, Director)$

european$(Director)$

review$(Title, Critique)$

GAV: associated to relations in the global schema we have views over the sources

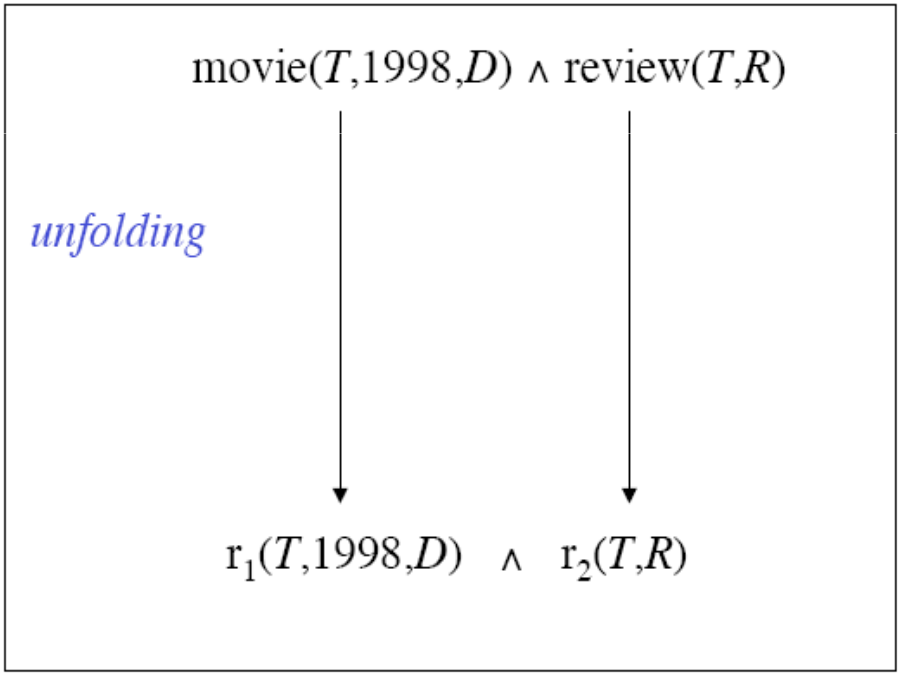
movie(T, Y, D) $\rightsquigarrow$ $\{ (T, Y, D) \mid \mathsf{r}_1(T, Y, D) \}$

european(D) $\rightsquigarrow$ $\{ (D) \mid \mathsf{r}_1(T, Y, D) \}$

review(T, R) $\rightsquigarrow$ $\{ (T, R) \mid \mathsf{r}_2(T, R) \}$

Example of query processing

The query $\{ (T, R) \mid \text{movie}(T, 1998, D) \wedge \text{review}(T, R) \}$ is processed by means of unfolding, i.e., by expanding each atom according to its associated definition in $\mathcal{M}$, so as to come up with source relations. In this case:



Gav & Lav: comparison

LAV: (Information Manifold, DWQ)

- Quality depends on how well we have characterized the sources
- High modularity and extensibility (if the global schema is well designed, when a source changes, only its definition is affected)
- Query processing needs reasoning (query answering complex)

GAV: (Carnot, SIMS, Tsimmis, IBIS, Momis, DisAtDis, . . .)

- Quality depends on how well we have compiled the sources into the global schema through the mapping
- Whenever a source changes or a new one is added, the global schema needs to be reconsidered
- Query processing can be based on some sort of unfolding (query answering looks easier – without constraints)

IMDb: mapping

A DL-Lite$_A$ ontology with mappings is characterized by a triple

O_m =<T,M,DB> such that:

- T is a DL-Lite$_A$ **TBox**
- DB is a relational database
- M is a set of mapping assertion, partitioned into two sets, M_t and M_a

Where:

M_t is a set of so-called ***typing mapping assertions***, each one of the form

$$\varphi \rightarrow T_i$$

where φ is a query of arity 1 over DB denoting the projection of one relation over one of its columns, and T_i is one of the DL-Lite$_A$ data types;

M_a is a set of ***data-to-object mapping assertions*** (or simply mapping assertions), each one of the form

$$\varphi \rightarrow \psi$$

where φ is an arbitrary SQL query of arity n>0 over DB, ψ is a conjunctive query over T of arity n'>0 without non distinguished variables, that possibly involves variable terms. A variable term is a term of the same form as the object terms introduced above, with the difference that variables appear as argument of the function. In other words, a variable terms has the form *f(z),* where *f* is a function symbol in Λ of arity m, and *z* denotes an m-tuple of variables.

Typing mapping assertion example

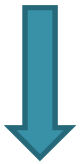
- $\mathbf{M_{t1}}$: SELECT id FROM ACTOR ⇝ xsd:int
- $\mathbf{M_{t2}}$: SELECT name FROM ACTOR ⇝ xsd:string
- $\mathbf{M_{t3}}$: SELECT surname FROM ACTOR ⇝ xsd:string
- $\mathbf{M_{t4}}$: SELECT sex FROM ACTOR ⇝ xsd:string
- $\mathbf{M_{t5}}$: SELECT actor FROM actor2movie ⇝ xsd:int

Data-to-object mapping assertions example

Λ(OBJECT VARIABLE TERMS**)={actor(x)}**

$\mathbf{M_{m1}}$: SELECT *ID,name,surname,sex*

FROM ACTOR



ACTOR(**actor**(*ID*)),
ID(**actor**(*ID*),*ID*),name(**actor**(*ID*),*name*),surname(**actor**(*ID*),*surname*),sex(**actor**(*ID*),*sex*)

Riferimenti

- www.imdb.com
- Dispense tratte dal corso di "seminario di ingegneria del software"
- Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati. Linking Data to Ontologies: The Description Logic DL-Lite$_A$. In *Proc. of the 2006 International Workshop on OWL: Experiences and directions (OWLED 2006).*
- Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati. MASTRO-I: Efficient integration of relational data through DL ontologies. In *Proc. of the 2007 Description Logic Workshop (DL 2007).*
- Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati. Linking Data to Ontologies.