Università degli Studi
di Roma "Sapienza"

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

anno accademico 2006-2007

Seminario di ingegneria del software

Autore

Cristiano Sticca

SOMMARIO

Introduzione	pag.3
Obiettivi	pag.4
Problematiche	pag.5
Struttura dei file importati dal sito www.imdb.com	pag.6
Parser Java	pag.13
Script SQL	pag.20
Schema ER	pag.22
Schema logico	pag.23
Schema fisico	pag.29
Schema fisico con vincoli di foreign key	pag.37
Ontologia: concetti generali	pag.49
Linguaggi per Ontologie: OWL	pag.50
Description Logics: concetti generali	pag.55
DL-Lite family: DL-Lite & DL-Lite	pag.56
DL-Lite	pag.61
Ontologia dell' IMDb	pag.64
Mapping: GAV & LAV	pag.79
IMDb mapping	pag.86
IMDb: data type mapping	pag.87
IMDb: data to object mapping	pag.93
Riferimenti	pag.105

Introduzione

Il lavoro che è stato fatto con questa tesina ha voluto illustrare come fare essenzialmente "data integration" relativamente alla parte sulle ontologie e sui mapping.
Scopo principale della tesina è stato quello di ricreare in ambiente locale ciò che è presente sul sito www.imdb.com , ovvero un database che rappresenta tutta la cinematografia dai suoi albori ad oggi.
Dopo il lavoro di creazione del DB, che ha impiegato gran parte del tempo e delle risorse, il compito si è concentrato nella stesura dell'ontologia scritta con una description logic, interamente studiata e realizzata nel Dipartimento di Informatica e Sistemistica della facoltà di Ingegneria dell' università "Sapienza" di Roma, vale a dire DL-Lite$_A$.
In seguito si è voluto creare un mapping tra l'ontologia appena descritta e il database locale.

Obiettivi

Gli obiettivi del presente lavoro sono stati essenzialmente:

- 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

Problematiche

I problemi maggiori si sono riscontrati nella prima parte della tesina, ovvero nel periodo in cui tutte le risorse sono state impiegate per creare il database locale.
Motivo di tali problemi è stato senza dubbio la grande quantità di file importati dal sito http://imdb.com/interfaces , in quanto questi ultimi hanno una formattazione pressoché diversa l'uno dall'altro e ciò ha richiesto quindi la realizzazione di moltissimi parser java il cui compito è stato quello di creare script SQL per il popolamento del database.
Ma il problema principale è stato senza ombra di dubbio il tempo impiegato per popolare il DB con i script sopra citati.

Struttura dei file importati dal sito www.imdb.com

Il sito come già detto in precedenza permette di poter effettuare il download di particolari file che rappresentano il contenuto informativo del database su internet. Vengono illustrati ora alcuni file importati dal sito.

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	
#Bfl O {ggGX = STwWcfl x 2s4 (1963)		1963
$ (1971)	1971	
$1,000 Reward (1913)		1913
$1,000 Reward (1915)		1915
$1,000 Reward (1923)		1923

actors.LIST

RC: 0x0A29EBA9 File: actors.list Date: Fri Sep 14 01:00:00 2007

COPYING POLICY: Internet Movie Database (IMDb)
==========

remain intact.

3. Specifically the files may NOT be used to construct any kind of on-line database (except for individual personal use). Clearance for ALL such on-line data resources must be requested from Internet Movie Database Ltd

4. In addition, copies of the Internet Movie Database frequently asked questions list and additions guide must be made available in the same area / by the same method as the other database files.

5. CD-ROM distribution is prohibited without written permission from the Internet Movie Database Ltd

Distribution by e-mail, BBS and Internet systems is positively encouraged within these limitations.

The files and software which make up the movie database may be uploaded to commercial BBS systems providing that the above conditions are met and no *additional* fees are applied above the standard connect time or downloading charges.

For further information contact <licensing@imdb.com>

Welcome to the latest version of the actors list.

Please feel free to submit entries for actors not already on the list,
or new entries for existing actors.

This list is managed by Giancarlo Cairella <actresses@imdb.com>; from
1990-1998 it was managed by Col Needham <col@imdb.com>.

If you have any additions or corrections please respond by e-mail only. The section at the end of the list contains details on the formats to use. The copying policy is also described at the end of the list.

The Internet Movie Database consists of the following lists:

List	Maintained by	Updated
Actors	Giancarlo Cairella http://imdb.com/contact	24-06-05
Actresses	Giancarlo Cairella http://imdb.com/contact	24-06-05
Alternative Names	Duncan Smith http://imdb.com/contact/	24-06-05
Alternative Titles	Michel Hafner http://imdb.com/contact/	17-06-05
Alternative Versions	Giancarlo Cairella http://imdb.com/contact	17-06-05
Biographies	Geoff Leonard http://imdb.com/contact/	24-06-05
Business	Giancarlo Cairella http://imdb.com/contact	17-06-05
Cast Completion	Giancarlo Cairella http://imdb.com/contact	17-06-05
Certificates	Peter Simeon http://imdb.com/contact/	24-06-05
Cinematographers	Michel Hafner http://imdb.com/contact/	17-06-05
Color Information	Mark Bailey http://imdb.com/contact/	24-06-05
Composers	Michel Hafner http://imdb.com/contact/	17-06-05
Costume Designers	Duncan Smith http://imdb.com/contact/	17-06-05

Countries | Peter Simeon http://imdb.com/contact/ | 17-06-05
Crazy Credits | Mark Bailey http://imdb.com/contact/ | 17-06-05
Crew Completion | Giancarlo Cairella http://imdb.com/contact| 17-06-05
Directors | Duncan Smith http://imdb.com/contact/ | 17-06-05
Distributors | Peter Simeon http://imdb.com/contact/ | 24-06-05
Editors | Duncan Smith http://imdb.com/contact/ | 17-06-05
Genres | Jake Dias http://imdb.com/contact/ | 24-06-05
Goofs | Col Needham http://imdb.com/contact/ | 24-06-05
Keywords | Jake Dias http://imdb.com/contact/ | 24-06-05
Languages | Peter Simeon http://imdb.com/contact/ | 24-06-05
Laser Discs | Peter Simeon http://imdb.com/contact/ | 07-21/00
Literature | Giancarlo Cairella http://imdb.com/contact| 17-06-05
Locations | Mark Bailey http://imdb.com/contact/ | 17-06-05
MPAA Ratings Reasons | Jon Reeves http://imdb.com/contact/ | 17-06-05
Misc. Companies | Mark Bailey http://imdb.com/contact/ | 24-06-05
Misc. Filmography | Peter Simeon http://imdb.com/contact/ | 24-06-05
Movie Links | Ron Higgins http://imdb.com/contact/ | 24-06-05
Movies | Michel Hafner http://imdb.com/contact/ | 17-06-05
Plot Summaries | Colin Tinto http://imdb.com/contact/ | 24-06-05
Producers | Andre Bernhardt http://imdb.com/contact/ | 17-06-05
Production Companies | Mark Bailey http://imdb.com/contact/ | 24-06-05
Production Designers | Duncan Smith http://imdb.com/contact/ | 17-06-05
Quotes | Col Needham http://imdb.com/contact/ | 24-06-05
Ratings | IMDb Helpdesk http://imdb.com/contact/ | 24-06-05
Release Dates | Mark Bailey http://imdb.com/contact/ | 24-06-05
Running Times | Mark Bailey http://imdb.com/contact/ | 24-06-05
SFX Companies | Mark Bailey http://imdb.com/contact/ | 24-06-05
Sound Mix | Mark Bailey http://imdb.com/contact/ | 24-06-05
Soundtracks | Ron Higgins http://imdb.com/contact/ | 24-06-05
Tag Lines | Mark Bailey http://imdb.com/contact/ | 24-06-05
Technical Info | Peter Simeon http://imdb.com/contact/ | 24-06-05
Trivia | Tim Norris http://imdb.com/contact/ | 24-06-05
Writers | Duncan Smith http://imdb.com/contact/ | 17-06-05
---

dd/mm/yy

SEARCHING THE DATABASE
==========

The movie database frequently asked questions list contains more information on the whole movie database project. For a copy send an e-mail message with the subject "HELP FAQ" to <mail-server@imdb.com>. Here is a summary of the ways to access the database:

(1) WWW interface

The Internet Movie Database is available over the WWW. The following sites are owned and operated by or for the IMDb:

http://us.imdb.com/ [USA]
http://uk.imdb.com/ [UK]

News and pointers to all IMDb sites are available at IMDb HQ:

http://www.imdb.com/

(2) e-mail interface

For details send a message with the subject HELP to <mail-server@imdb.com>

(3) local installation (Unix/Amiga)

The movie database package enables you to install the data locally and provides a variety of search tools. It is available via anonymous FTP:

uiarchive.cso.uiuc.edu in /pub/info/imdb/tools/moviedb-3.4a.tar.gz

ftp.funet.fi in /pub/culture/tv+film/database/tools/moviedb-3.4a.tar.gz

ftp.fu-berlin.de in /pub/misc/movies/database/tools/moviedb-3.4a.tar.gz

ftp.sunet.se in /pub/tv+movies/imdb/tools/moviedb-3.4a.tar.gz

see the README file in the same directories for more information. The Amiga version is in the file imdb3_5_Amiga.lha

(4) local installation (OS/2)

The Alternative Movie Database package provides a graphical and text based interface for OS/2:

uiarchive.cso.uiuc.edu in /pub/info/imdb/tools/os2/

ftp.funet.fi in /pub/culture/tv+film/database/tools/os2/

ftp.fu-berlin.de in pub/misc/movies/database/tools/os2/

ftp.sunet.se in /pub/tv+movies/imdb/tools/os2/

(5) local installation (Windows 9x/Windows NT)

The Alternative Movie Database package is also available as a text only interface for Win-32 systems (9x/NT):

uiarchive.cso.uiuc.edu in /pub/info/imdb/tools/w32/

ftp.funet.fi in /pub/culture/tv+film/database/tools/w32/

ftp.fu-berlin.de in pub/misc/movies/database/tools/w32/

ftp.sunet.se in /pub/tv+movies/imdb/tools/w32/

RULES:

1 Movies and recurring TV roles only, no TV guest appearances
2 Please submit entries in the format outlined at the end of the list
3 Feel free to submit new actors

"xxxxx" = a television series
"xxxxx" (mini) = a television mini-series
[xxxxx] = character name
<xx> = number to indicate billing position in credits
(TV) = TV movie, or made for cable movie

(V) = made for video movie (this category does NOT include TV episodes repackaged for video, guest appearances in variety/comedy specials released on video, or self-help/physical fitness videos)

THE ACTORS LIST
==========

Name Titles
---- ------
$, Steve E.R. Sluts (2003) (V) <12>

'babeepower' Viera, Michael Rock Steady (2002) [Stevie]
"Lyricist Lounge Show, The" (2000) [Various/lyricist]

'Cartucho' Pena, Ramon Natas es Satan (1977) [Nigth Club Owner]

'Chincheta', Eloy ¡Ja me maaten...! (2000) [Gitano 1] <20>

'El de Chipiona', Antonio Guitarra muda, La (1953) [Himself]

'El Francés', José Alma gitana (1996) <45>
Premios Amigo 2000 (2000) (TV) [Himself]
Que dis que din, O (2003) [Himself]
"Al salir de clase" (1997) {Por una buena causa (#6.4)} [Himself]
"Música uno" (2004) {(2004-05-08)} [Himself]

'El Gato', Félix Noche de los inocentes, La (1997) (TV) [Himself]
Pelotazo nacional (1993) <12>
"Esto es espectáculo" (1994) {(1994-12-23)}
"Noche de fiesta" (1999) {(2003-07-12)}
"Noche de fiesta" (1999) {(2003-08-16)}
"Querida Concha" (1992)
"Uno para todas" (1995) {(1996-02-13)} [Staff Humorist] <3>
"Uno para todas" (1995) {(1996-02-20)} [Staff Humorist] <3>
"Uno para todas" (1995) {(1996-02-27)} [Staff Humorist] <3>
"VIP noche" (1990) {(1990-11-25)}

times.LIST

```
RUNNING TIMES LIST
==================
0016643225059 (1994)                                    Singapore:5       (Singapore International Film Festival)
002 agenti segretissimi (1964)                          83
002 operazione Luna (1965)                              90
"003 y medio" (1979)                                    Spain:160
007 in Egypt (2006) (V)                                 UK:6
007 in Rio (1979)                               UK:13
007: Licence to Restore (2006) (V)              USA:13
007 Stage Dedication (1977)                             UK:2
007: The Return (1995) (TV)                             UK:60
008 (2003)                                              USA:7
008 - Agent wider Willen (2000) (TV)                    93
00h17 (2005)                                            France:8
00 Schneider - Jagd auf Nihil Baxter (1994)     90
00Sex, es ist niemals zu spät! (1998) (V)       Germany:115
011 Beograd (2003)                                      83
[0-1] (2003)                                            15        (original version)
01412 pasasingeum (2000)                                95
01-99 (1959)                                            22
0!1 (Zero Bang One) (2000)                              USA:20
02 Wireless Festival (2007) (TV)                UK:48
03/02/05 (2005) (V)                                     80
:03 from Gold (2002) (TV)                       59
"0416-os szökevény, A" (1970) (mini)                    Hungary:254
04:44 (2005)                                            5
'04: How Was It for You? (2005) (TV)                    Ireland:65        (including commercials)
0506HK (2007)                                           63
0567 - Appunti per un documentario su Pozzuoli (1987)   Germany:83
05h42 (2003)                                            South Africa:42
'05: How Was It for You? (2005) (TV)                    Ireland:60        (including commercials)
0-600-Amor ya (1998)                                    Argentina:25
06/05 (2004)                                            Netherlands:117
06 (1994)                                               87
0623-ZN (Sentenciados) (2003)                           7
'06: The Big One (2006)                                 USA:6
```

Da come è facile notare questi file, come già detto prima, sono strutturati in maniera diversa l'uno dall'altro e questo ha portato dunque alla realizzazione di molteplici parser che descriviamo adesso.

Parser Java

Vediamo la struttura ora di qualche parser. Più precisamente analizziamo la struttura dei parser adibiti alla realizzazione degli script SQL per l'inserimento di movie e actors (i file presentati nella sezione precedente).

movies.java

```
import java.sql.*;
import java.net.*;
import java.io.*;
import java.lang.*;
import java.util.*;

public class movies {


        public static void main (String[] args) throws Exception
            {



        String primo,secondo;
                    secondo = null;
                    int movieid=0;
                    String[]res;

        //script SQL
        FileWriter fi = new FileWriter("G:\\Documents and Settings\\cristiano\\Desktop\\tesina seminario\\tesina\\finalScripts\\movies.sql");
        PrintWriter out=new PrintWriter(fi);

        //file sorgente da cui estrapolare i "movie"
        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();

     //inserimento su DB… è stata fatta la scelta di utilizzare I "multiple inserts" molto più efficienti
    out.print("INSERT IGNORE INTO movies VALUES\n");


         while (nextStr!=null){

                    if (nextStr.equals("------------------------------------------------------------------------------")){
        System.out.println("Movies inseriti nel db");
        break;
    }


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

```
//da qui in poi inizia il vero  e proprio parsing
    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);
                        //      primo=primo.replace("\"","");
                        }

//inserimento dei “movie”
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.java

```
import java.sql.*;
import java.net.*;
import java.io.*;
import java.lang.*;
import java.util.*;

public class actors {


        public static void main (String[] args) throws Exception
    {



        String primo,secondo,as;
        primo = null;
        secondo = null;
        as = null;
        String[]res;
        int inizio=0;
        int inizio2=0;
        String nome = null;
    String cognome = null;
    String year=null;
    String[] arr=null;
    int attoreid=1;
    int movieid=0;
    String nomecognome=null;
```

//script SQL per gli “actors”

```
 FileWriter fi = new FileWriter("G:\\Documents and Settings\\cristiano\\Desktop\\tesina seminario\\tesina\\finalScripts\\actors.sql");
PrintWriter out=new PrintWriter(fi);
```

//script SQL per la relazione “actor2movie” ovvero gli attori con i relativi movie

```
FileWriter fi3 = new FileWriter("G:\\Documents and Settings\\cristiano\\Desktop\\tesina seminario\\tesina\\finalScripts\\actors2movie.sql");
PrintWriter out3=new PrintWriter(fi3);
```

//file sorgente da cui estrapolare gli “actors”

```
FileReader f = new FileReader("G:\\Documents and Settings\\cristiano\\Desktop\\tesina seminario\\tesina\\file IMDB\\actors.txt");
BufferedReader filebuf = new BufferedReader(f);
```

//connessione al database necessaria per poter individuare l’ID del movie

```
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");
    }




    out.print("INSERT IGNORE INTO actor VALUES\n");
```

```
out3.print("INSERT IGNORE INTO actor2movie VALUES\n");
```

//inizio parsing

```
 String nextStr;
nextStr = filebuf.readLine();

                    if (args[0].equals("-") && (args[1].startsWith("*"))){

                              String num=args[1].substring(1);
                              attoreid=Integer.parseInt(num);

                     while (!nextStr.equals("THE ACTORS LIST")) {
                              nextStr=filebuf.readLine();
                    }

                    nextStr = filebuf.readLine();
nextStr = filebuf.readLine();
nextStr = filebuf.readLine();
nextStr = filebuf.readLine();
nextStr = filebuf.readLine();

                    }

                    if (!args[0].equals("-")){


                    while (!nextStr.startsWith(args[0])) {
                              nextStr=filebuf.readLine();
                              }
                              attoreid=Integer.parseInt(args[1]);
                    }




        while (nextStr!=null){

                                        try{


                     if (nextStr.equals("SUBMITTING UPDATES")){
                              System.out.println("Actors 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);

            }

            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;


            }
```

//inserimento nello script relativo ad “actors”

```
            out.append("('"+attoreid+"','"+nome+"','"+cognome+"','M'),\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{


                                        //prelievo ID del movie dal database

                                        Statement s2 = conn2.createStatement ();

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

                                        rs3.last();

                                movieid =(Integer)rs3.getObject(1);

                                        s2.close ();


}
catch(Exception e){
}


                                        //inserimento nello script relativo alla relazione “actor2movie”

                                        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();



  }



            }
```

Script SQL

Di seguito vengono mostrati gli script SQL generati dai parser di cui si è parlato nella sezione precedente.

movies.sql

```
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),
```

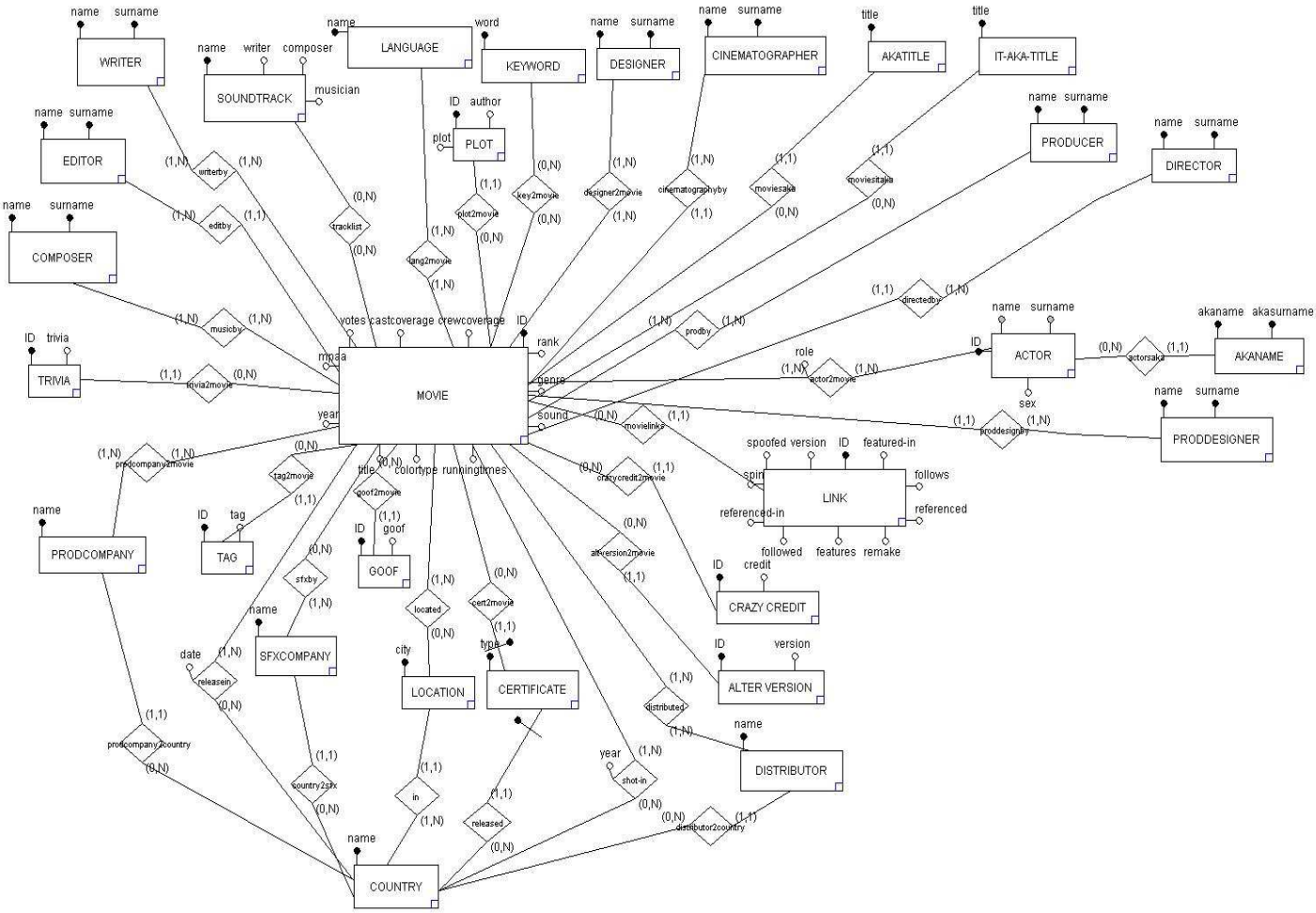
actors.sql

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','AAli taleb','F'),('22','Mrs.','ACosta','F'),('23','Michele','ACourt','F'),

actor2movie.sql

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
PRODCOMPANY
TAG
GOOF
LINK
CRAZY CREDIT
SFXCOMPANY
LOCATION
CERTIFICATE
ALTER VERSION
DISTRIBUTOR
COUNTRY

Schema logico

key2movie(MOVIE,KEYWORD)

FK: key2movie[MOVIE] ⊆ MOVIES[ID]
FK:key2movie[KEYWORD] ⊆ KEYWORD[word]

plot2movie(MOVIE,PLOT)

FK: plot2movie[MOVIE] ⊆ MOVIES[ID]
FK:plot2movie[PLOT] ⊆ PLOT[ID]

goof2movie(MOVIE,GOOF)

FK: goof2movie[MOVIE] ⊆ MOVIES[ID]
FK:goof2movie[GOOF] ⊆ GOOF[ID]

altver2movie(MOVIE,ALTVERSION)

FK:altver2movie[MOVIE] ⊆ MOVIES[ID]
FK:altver2movie[ALTVERSION] ⊆ ALTVERSION[ID]

trivia2movie(MOVIE,TRIVIA)

FK: trivia2movie[MOVIE] ⊆ MOVIES[ID]
FK:trivia2movie[TRIVIA] ⊆ TRIVIA[ID]

tag2movie(MOVIE,TAG)

FK: tag2movie[MOVIE] ⊆ MOVIES[ID]
FK:tag2movie[TAG] ⊆ TAG[ID]

crazy2movie(MOVIE,CRAZYCREDIT)

FK: crazy2movie[MOVIE] ⊆ MOVIES[ID]
FK:crazy2movie[CRAZYCREDIT] ⊆ CRAZYCREDIT[ID]

lang2movie(MOVIE,LANGUAGE)

FK: lang2movie[MOVIE] ⊆ MOVIES[ID]
FK:lang2movie[LANGUAGE] ⊆ LANGUAGE[name]

movielinks(MOVIE,LINK)

FK: movielinks[MOVIE] ⊆ MOVIES[ID]
FK:movielinks[LINK] ⊆ LINK[ID]

KEYWORD(word)

PLOT(ID,plot,author)

FK: PLOT[ID] ⊆ plot2movie[PLOT]

GOOF(ID,goof)

FK: GOOF[ID] ⊆ goof2movie[GOOF]

ALTVERSION(ID,version)

FK: ALTVERSION[ID] ⊆ altversion2movie[ALTVERSION]

TRIVIA(ID,trivia)

FK: TRIVIA[ID] ⊆ trivia2movie[TRIVIA]

TAG(ID,tag)

FK: TAG[ID] ⊆ tag2movie[TAG]

CRAZYCREDIT(ID,credit)

FK: CRAZYCREDIT[ID] ⊆ crazycredit2movie[CRAZYCREDIT]

LANGUAGE(name)

inclusione: LANGUAGE[name] ⊆ lang2movie[LANGUAGE]

musicby(COMPOSERN,COMPOSERS,MOVIE)

FK: musicby[COMPOSERN,COMPOSERS] ⊆ COMPOSER[name,surname]
FK:musicby[MOVIE] ⊆ MOVIES[ID]

COMPOSER(name,surname)

inclusione: COMPOSER[name,surname] ⊆ musicby[COMPOSERN,COMPOSERS]

EDITOR(name,surname)

inclusione: EDITOR[name,surname] ⊆ editbyby[EDITORN,EDITORS]

writerby(WRITERN,WRITERS,MOVIE)

FK: writerby[WRITERN,WRITERS] ⊆ WRITER[name,surname]
FK:writerby[MOVIE] ⊆ MOVIES[ID]

WRITER(name,surname)

inclusione: WRITER[name,surname] ⊆ writerby[WRITERN,WRITERS]

tracklist(SOUNDTRACK,MOVIE)

FK:tracklist[SOUNDTRACK] ⊆ SOUNDTRACK[name]
FK:tracklist[MOVIE] ⊆ MOVIES[ID]

LINK(ID,featuredin,referencedin,spin,spoofed,version,followed,features,remake,follows,referencs)

FK: LINK[ID] ⊆ movielinks[LINK]

SOUNDTRACK(name,writer,composer,singer)

prodcompany2movie(PRODCOMPANY,MOVIE)

FK: prodcompany2movie[PRODCOMPANY] ⊆ PRODCOMPANY[name]
FK: prodcompany2movie[MOVIE] ⊆ MOVIES[ID]

actors2movie(ACTOR,MOVIE,role)

FK: actor2movie[ACTOR] ⊆ ACTOR[ID]
FK: actor2movie[MOVIE] ⊆ MOVIES[ID]

ACTOR(ID,name,surname,sex)

chiave: name,surname
inclusione: ACTOR[ID] ⊆ actor2movie[ACTOR]

AKANAME(akaname,akasurname)

FK: AKANAME[akaname,akasurname] ⊆ actorsaka[AKANAME,AKASURNAME]

PRODDESIGNER(name,surname)

inclusione: PRODDESIGNER[name,surname] ⊆ proddesignby[PRODDESIGNERN,PRODDESIGNERS]

DIRECTOR(name,surname)

inclusione: DIRECTOR[name,surname] ⊆ directedby[DIRECTORN,DIRECTORS]

prodby(PRODUCERN,PRODUCERS,MOVIE)

FK: prodby[PRODUCERN,PRODUCERS] ⊆ PRODUCER[name,surname]
FK: prodby[MOVIE] ⊆ MOVIES[ID]

PRODUCER(name,surname)

inclusione: PRODUCER[name,surname] ⊆ prodby[PRODUCERN,PRODUCERS]

prodcompany2country(PRODCOMPANY,COUNTRY)

FK: prodcompany2country[PRODCOMPANY] ⊆ PRODCOMPANY[name]
FK: prodcompany2country[COUNTRY] ⊆ COUNTRY[name]

AKATITLE(title)

FK: AKATITLE[title] ⊆ moviesaka[AKATITLE]

ITAKATITLE(title)

FK:IT AKATITLE[title] ⊆ moviesitaka[ITAKATITLE]

CINEMATOGRAPHER(name,surname)

inclusione: CINEMATOGRAPHER[name,surname] ⊆ cinematographer-by[CINEMATOGRAPHERN,CINEMATOGRAPHERS]

designer2movie(DESIGNERN,DESIGNERS,MOVIE)

FK: designer2movie[DESIGNERN,DESIGNERS] ⊆ DESIGNER[name,surname]
FK: designer2movie[MOVIE] ⊆ MOVIES[ID]

DESIGNER(name,surname)

Inclusione: DESIGNER[name,surname] ⊆ designer2movie[DESIGNERN,DESIGNERS]

distributed(DISTRIBUTOR,MOVIE)

FK: distributed[DISTRIBUTOR] ⊆ DISTRIBUTOR[name]
FK: distributed[MOVIE] ⊆ MOVIES[ID]

DISTRIBUTOR(name)

inclusione: DISTRIBUTOR[name] ⊆ distributed[DISTRIBUTOR]
FK: DISTRIBUTOR[name] ⊆ distributor2country[DISTRIBUTOR]

sfxby(SFXCOMPANY,MOVIE)

FK: sfxby[SFXCOMPANY] ⊆ SFXCOMPANY[name]
FK: sfxby[MOVIE] ⊆ MOVIES[ID]

SFXCOMPANY(name)

inclusione: SFXCOMPANY[name] ⊆ sfxby[SFXCOMPANY]
FK: SFXCOMPANY[name] ⊆ country2sfx[SFXCOMPANY]

actorsaka(ACTOR,AKANAME,AKASURNAME)

FK: actorsaka[ACTOR] ⊆ ACTOR[ID]
FK: actorsaka[AKANAME,AKASURNAME] ⊆ AKANAME[name,surname]

located(LOCATIONCITY,LOCATIONCOUNTRY,MOVIE)

FK: located[LOCATIONCITY] ⊆ LOCATION[city]
FK: located[LOCATIONCOUNTRY] ⊆ COUNTRY[name]
FK: located[MOVIE] ⊆ MOVIES[ID]

LOCATION(city,COUNTRY)

FK: LOCATION[COUNTRY] ⊆ COUNTRY[name]

shotin(COUNTRY,MOVIE,year)

FK: shotin[COUNTRY] ⊆ COUNTRY[name]
FK: shotin[MOVIE] ⊆ MOVIES[ID]

COUNTRY(name)

inclusione: COUNTRY[name] ⊆ releasein[COUNTRY]

CERTIFICATE(type,COUNTRY,MOVIE)

FK: CERTIFICATE[COUNTRY] ⊆ COUNTRY[name]
FK: CERTIFICATE[MOVIE] ⊆ MOVIES[ID]

editby(MOVIE,EDITORN,EDITORS)

FK: editby[EDITORN,EDITORS] ⊆ EDITOR[name,surname]
FK: editby[MOVIE] ⊆ MOVIES[ID]

proddesignby(MOVIE,PRODDESIGNERN,PRODDESIGNERS)

FK: proddesignby[PRODDESIGNERN,PRODDESIGNERS] ⊆ PRODDESIGNER[name,surname]
FK: proddesignby[MOVIE] ⊆ MOVIES[ID]

directedby(MOVIE,DIRECTORN,DIRECTORS)

FK: directedby[DIRECTORN,DIRECTORS] ⊆ DIRECTOR[name,surname]
FK: directedby[MOVIE] ⊆ MOVIES[ID]

cinematographyby(MOVIE,CINEMATOGRAPHERN,CINEMATOGRAPHERS)

FK: cinematographyby[CINEMATOGRAPHERN, CINEMATOGRAPHERS] ⊆ CINEMATOGRAPHER[name,surname]
FK: cinematographyby[MOVIE] ⊆ MOVIES[ID]

releasein(MOVIE,COUNTRY,date)

FK: releasein[COUNTRY] ⊆ COUNTRY[name]
FK: releasein[MOVIE] ⊆ MOVIES[ID]

moviesaka(AKATITLE, MOVIE)

FK: moviesaka[AKATITLE] ⊆ AKATITLE[title]
FK: moviesaka[MOVIE] ⊆ MOVIES[ID]

moviesitaka(ITAKATITLE, MOVIE)

FK: moviesitaka[ITAKATITLE] ⊆ ITAKATITLE[title]
FK: moviesitaka[MOVIE] ⊆ MOVIES[ID]

distributor2country(DISTRIBUTOR, COUNTRY)

FK: distributor2country[DISTRIBUTOR] ⊆ DISTRIBUTOR[name]
FK: distributor2country[COUNTRY] ⊆ COUNTRY[name]

country2sfx(SFXCOMPANY, COUNTRY)

FK: country2sfx[SFXCOMPANY] ⊆ SFXCOMPANY[name]
FK: country2sfx[COUNTRY] ⊆ COUNTRY[name]

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

```
drop database if exists imdb;

create database imdb;

use imdb;

drop table if exists keyword;

create table keyword (
        word varchar(200) primary key
);

drop table if exists key2movie;

create table key2movie (
        movie int(8),
        keyword varchar(200),
        primary key (movie,keyword)
        );

drop table if exists plot;

create table plot (
        id int(8) primary key,
        plot longtext,
        author varchar(200)
);

drop table if exists plot2movie;

create table plot2movie (
        plot int(8) primary key,
        movie int(8)
);


drop table if exists goof;

create table goof (
        id int(8) primary key,
        goof longtext
);

drop table if exists goof2movie;

create table goof2movie (
        goof int(8) primary key,
        movie int(8)
);


drop table if exists altversion;

create table altversion (
        id int(8) primary key,
        version longtext
);

drop table if exists altversion2movie;

create table altversion2movie (
```

```
	altversion int(8) primary key,
	movie int(8)
);

drop table if exists trivia;

create table trivia (
	id int(8) primary key,
	trivia longtext
);

drop table if exists trivia2movies;

create table trivia2movies (
	trivia int(8) primary key,
	movie int(8)
);

drop table if exists tag;

create table tag (
	id int(8) primary key,
	tag longtext
);

drop table if exists tag2movie;

create table tag2movie (
	tag int(8) primary key,
	movie int(8)

);



drop table if exists crazycredit;

create table crazycredit (
	id int(8) primary key,
	credit longtext
);

drop table if exists crazycredit2movie;

create table crazycredit2movie (
	credit int(8) primary key,
	movie int(8)
);


drop table if exists language;

create table language (
	name varchar(200) primary key,
	check (name in (select language from lang2movie))
);

drop table if exists lang2movie;

create table lang2movie (
	movie int(8),
	language varchar(200),
	primary key(movie,language)
);
```

```
drop table if exists composer;

create table composer (
	name varchar(200),
	surname varchar(200),
	primary key (name,surname),
	check (name,surname in (select composerN,composerS from musicby))
);

drop table if exists musicby;

create table musicby (
	composerN varchar(200),
	composerS varchar(200),
	movie int(8),
	primary key(composerN,composerS,movie)

);


drop table if exists editby;

create table editby (
	movie int(8) primary key,
	editorN varchar(200),
	editorS varchar(200)
);


drop table if exists editor;

create table editor (
	name varchar(200),
	surname varchar(200),
	primary key (name,surname),
	check (name,surname in (select editorN,editorS from editby))
);


drop table if exists writer;

create table writer (
	name varchar(200),
	surname varchar(200),
	primary key (name,surname),
	check (name,surname in (select writerN,writerS from writerby))
);

drop table if exists writerby;

create table writerby (
	writerN varchar(200),
	writerS varchar(200),
	movie int(8),
	primary key (writerN,writerS,movie)
);

drop table if exists soundtrack;

create table soundtrack (
	name varchar(200) primary key,
	writer varchar(200),
	composer varchar(200),
	musician varchar(200)
```

```
);

drop table if exists tracklist;

create table tracklist (
	soundtrack varchar(200),
	movie int(8),
	primary key (soundtrack,movie)
);

drop table if exists prodcompany;

create table prodcompany (
	name varchar(200) primary key,
	check (name in (select prodcompany from prodcompany2movie))
);

drop table if exists prodcompany2movie;

create table prodcompany2movie (
	prodcompany varchar(200),
	movie int(8),
	primary key (prodcompany,movie)
);

drop table if exists actor;

create table actor (
	id int(8) primary key,
	name varchar(200),
	surname varchar(200),
	sex enum('M','F'),
	unique(name,surname),
	index(name,surname),
	check (id in (select actor from actor2movie))
);

drop table if exists actor2movie;

create table actor2movie (
	actor int(8),
	movie int(8),
	role varchar(200),
	primary key(actor,movie)
);

drop table if exists akaname;

create table akaname (
	akaname varchar(200),
	akasurname varchar(200),
	primary key (akaname,akasurname)
);

drop table if exists actorsAka;

create table actorsAka (
	actor int(8),
	akaname varchar(200),
	akasurname varchar(200),
	primary key (akaname,akasurname)
);

drop table if exists proddesigner;
```

```
create table proddesigner (
	name varchar(200),
	surname varchar(200),
	primary key(name,surname),
	check (name,surname in (select proddesignerN,proddesignerS from proddesignby))
);


drop table if exists proddesignby;

create table proddesignby (
	movie int(8) primary key,
	proddesignerN varchar(200),
	proddesignerS varchar(200)
);


drop table if exists director;

create table director (
	name varchar(200),
	surname varchar(200),
	primary key (name,surname),
	check (name,surname in (select directorN,directorS from directedby))
);


drop table if exists directedby;

create table directedby (
	movie int(8) primary key,
	directorN varchar(200),
	directorS varchar(200)
);


drop table if exists producer;

create table producer (
	name varchar(200),
	surname varchar(200),
	primary key (name,surname),
	check (name,surname in (select producerN,producerS from prodby))
);

drop table if exists prodby;

create table prodby (
	producerN varchar(200),
	producerS varchar(200),
	movie int(8),
	primary key (producerN,producerS,movie)
);

drop table if exists akatitle;

create table akatitle (
	title varchar(200) primary key
);

drop table if exists itakatitle;

create table itakatitle (
	title varchar(200) primary key
);
```

```
drop table if exists cinematographer;

create table cinematographer (
        name varchar(200),
        surname varchar(200),
        primary key (name,surname),
        check (name,surname in (select cinematographerN,cinematographerS from cinematographyby))
);


drop table if exists cinematographyby;

create table cinematographyby (
        movie int(8) primary key,
        cinematographerN varchar(200),
        cinematographerS varchar(200)
);



drop table if exists designer;

create table designer (
        name varchar(200),
        surname varchar(200),
        primary key (name,surname),
        check (name,surname in (select designerN,designerS from designer2movie))
);


drop table if exists designer2movie;

create table designer2movie (
        designerN varchar(200),
        designerS varchar(200),
        movie int(8),
        primary key (designerN,designerS,movie)
);

drop table if exists distributor;

create table distributor (
        name varchar(200) primary key,
        check (name in (select distributor from distributed))
);

drop table if exists distributed;

create table distributed (
        distributor varchar(200),
        movie int(8),
        primary key (distributor,movie)
        );

drop table if exists sfxcompany;

create table sfxcompany (
        name varchar(200) primary key,
        check (name in (select sfxcompany from sfxby))
);

drop table if exists sfxby;
```

```
create table sfxby (
	sfxcompany varchar(200),
	movie int(8),
	primary key (sfxcompany,movie)
);

drop table if exists location;

create table location (
	city varchar(200),
	country varchar(200),
	primary key(city,country)
);

drop table if exists located;

create table located (
	locationcity varchar(200),
	locationcountry varchar(200),
	movie int(8),
	primary key(locationcity,locationcountry,movie)
	);

drop table if exists country;

create table country (
	name varchar(200) primary key,
	check (name in (select country from location))
);

drop table if exists shotin;

create table shotin (
	country varchar(200),
	movie int(8),
	year varchar(100),
	primary key (country,movie)
);

drop table if exists certificate;

create table certificate (
	country varchar(200),
	movie int(8),
	type varchar(100),
	primary key(country,movie,type)
);

drop table if exists movies;

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 lang2movie)),
        check (id in (select movie from writerby)),
        check (id in (select movie from prodcompany2movie)),
        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))

);


drop table if exists moviesAka;

create table moviesAka (
        akatitle varchar(200) primary key,
        movie int(8)
);

drop table if exists moviesItaka;

create table moviesItaka (
        itakatitle varchar(200) primary key,
        movie int(8)
);

drop table if exists distributor2country;

create table distributor2country (
        distributor varchar(200) primary key,
        country varchar(200)
);

drop table if exists country2sfx;

create table country2sfx (
        sfxcompany varchar(200) primary key,
        country varchar(200)
);


drop table if exists prodcompany2country;

create table prodcompany2country (
        prodcompany varchar(200) primary key,
        country varchar(200)
);

drop table if exists releasein;

create table releasein (
        movie int(8),
        country varchar(200),
        primary key(movie,country),
        date varchar(200)
);

drop table if exists link;

create table link (
        id int(8) primary key,
```

```
        featured_in varchar(200),
        referenced_in varchar(200),
        spin varchar(200),
        spoofed varchar(200),
        version varchar(200),
        followed varchar(200),
        features varchar(200),
        remake varchar(200),
        follows varchar(200),
        ref varchar(200)
);

drop table if exists movieLinks;

create table movieLinks (
        link int(8) primary key,
        movie int(8)
);
```

Schema fisico con vincoli di chiave esterna

```
alter table key2movie
add foreign key (movie)
references movies(id);

alter table key2movie
add foreign key (keyword)
references keyword(word);

alter table plot2movie
add foreign key (movie)
references movies(id);

alter table plot2movie
add foreign key (plot)
references plot(id);

alter table movieLinks
add foreign key (link)
references link(id);

alter table goof2movie
add foreign key (movie)
references movies(id);

alter table goof2movie
add foreign key (goof)
references goof(id);
```

```
alter table altversion2movie
add foreign key (movie)
references movies(id);

alter table altversion2movie
add foreign key (altversion)
references altversion(id);


alter table trivia2movies
add foreign key (movie)
references movies(id);

alter table trivia2movies
add foreign key (trivia)
references trivia(id);

alter table tag2movie
add foreign key (movie)
references movies(id);

alter table tag2movie
add foreign key (tag)
references tag(id);

alter table crazycredit2movie
add foreign key (movie)
references movies(id);

alter table crazycredit2movie
add foreign key (credit)
references crazycredit(id);
```

```
alter table lang2movie
add foreign key (movie)
references movies(id);

alter table lang2movie
add foreign key (language)
references language(name);

alter table plot
add foreign key (id)
references plot2movie(plot);

alter table link
add foreign key (id)
references movieLinks (link);

alter table goof
add foreign key (id)
references goof2movie(goof);

alter table altversion
add foreign key (id)
references altversion2movie(altversion);

alter table trivia
add foreign key (id)
references trivia2movies(trivia);

alter table crazycredit
add foreign key (id)
references crazycredit2movie(credit);
```

```
alter table musicby
add foreign key (composerN,composerS)
references composer(name,surname);

alter table musicby
add foreign key (movie)
references movies(id);

alter table writerby
add foreign key (writerN,writerS)
references writer(name,surname);

alter table writerby
add foreign key (movie)
references movies(id);

alter table tracklist
add foreign key (soundtrack)
references soundtrack(name);

alter table tracklist
add foreign key (movie)
references movies(id);

alter table prodcompany2movie
add foreign key (prodcompany)
references prodcompany(name);

alter table prodcompany2movie
add foreign key (movie)
references movies(id);
```

```
alter table prodcompany
add foreign key (name)
references prodcompany2country(prodcompany);

alter table actor2movie
add foreign key (movie)
references movies(id);

alter table actor2movie
add foreign key (actor)
references actor(id);

alter table akaname
add foreign key (akaname,akasurname)
references actorsAka(akaname,akasurname);

alter table actorsAka
add foreign key (actor)
references actor(id);

alter table actorsAka
add foreign key (akaname,akasurname)
references akaname(akaname,akasurname);

alter table prodby
add foreign key (producerN,producerS)
references producer(name,surname);

alter table prodby
add foreign key (movie)
references movies(id);
```

```
alter table prodcompany2country
add foreign key (prodcompany)
references prodcompany(name);

alter table prodcompany2country
add foreign key (country)
references country(name);

alter table akatitle
add foreign key (title)
references moviesaka(akatitle);

alter table itakatitle
add foreign key (title)
references moviesItaka(itakatitle);

alter table designer2movie
add foreign key (designerN,designerS)
references designer(name,surname);

alter table designer2movie
add foreign key (movie)
references movies(id);

alter table distributed
add foreign key (movie)
references movies(id);

alter table distributed
add foreign key (distributor)
references distributor(name);
```

```
alter table distributor
add foreign key (name)
references distributor2country(distributor);

alter table sfxby
add foreign key (sfxcompany)
references sfxcompany(name);

alter table sfxby
add foreign key (movie)
references movies(id);

alter table sfxcompany
add foreign key (name)
references country2sfx(sfxcompany);

alter table located
add foreign key (locationcity)
references location(city);

alter table located
add foreign key (locationcountry)
references country(name);

alter table located
add foreign key (movie)
references movies(id);
```

```
alter table location
add foreign key (country)
references country(name);

alter table shotin
add foreign key (country)
references country(name);

alter table shotin
add foreign key (movie)
references movies(id);

alter table certificate
add foreign key (movie)
references movies(id);

alter table certificate
add foreign key (country)
references country(name);

alter table editby
add foreign key (movie)
references movies(id);

alter table editby
add foreign key (editorN,editorS)
references editor(name,surname);

alter table proddesignby
add foreign key (movie)
references movies(id);
```

```
alter table proddesignby
add foreign key (proddesignerN,proddesignerS)
references proddesigner(name,surname);

alter table directedby
add foreign key (movie)
references movies(id);

alter table directedby
add foreign key (directorN,directorS)
references director(name,surname);

alter table cinematographyby
add foreign key (movie)
references movies(id);

alter table cinematographyby
add foreign key (cinematographerN,cinematographerS)
references cinematographer(name,surname);

alter table releasein
add foreign key (movie)
references movies(id);

alter table releasein
add foreign key (country)
references country(name);

alter table movies
add foreign key (id)
references proddesignby(movie);
```

```
alter table movies
add foreign key (id)
references directedby(movie);

alter table movies
add foreign key (id)
references editby(movie);

alter table movies
add foreign key (id)
references cinematographyby(movie);


alter table moviesAka
add foreign key (akatitle)
references akatitle(title);

alter table moviesAka
add foreign key (movie)
references movies(id);


alter table moviesItaka
add foreign key (itakatitle)
references itakatitle(title);


alter table moviesItaka
add foreign key (movie)
references movies(id);
```

```
alter table distributor2country
add foreign key (distributor)
references distributor(name);

alter table distributor2country
add foreign key (country)
references country(name);

alter table country2sfx
add foreign key (sfxcompany)
references sfxcompany(name);

alter table country2sfx
add foreign key (country)
references country(name);
```

Ontology

In both computer science and information science, an **ontology** is a data model that represents a set of concepts within a domain and the relationships between those concepts. It is used to reason about the objects within that domain.

Ontologies generally describe:

- **Individuals**: the basic or "ground level" objects
- **Classes**: sets, collections, or types of objects
- **Attributes**: properties, features, characteristics, or parameters that objects can have and share
- **Relations**: ways that objects can be related to one another
- **Events**: the changing of attributes or relations

Languages for ontologies

An ontology language is a formal language used to encode the ontology. There are a number of such languages for ontologies, both proprietary and standards-based:

- **OWL** is a language for making ontological statements, developed as a follow-on from RDF and RDFS. OWL is intended to be used over the World Wide Web, and all its elements (classes, properties and individuals) are defined as RDF resources, and identified by URIs.

Three Variants of OWL

- OWL Full
  - an extension of RDF
  - allows for classes as instances, modification of RDF and OWL vocabularies
- OWL DL
  - the part of OWL Full that fits in the Description Logic framework
  - known to have decidable reasoning
- OWL Lite
  - a subset of OWL DL
  - easier for frame-based tools to transition to
  - easier reasoning

How is OWL Used

1. build an ontology
   - create the ontology
   - name classes and provide information about them
   - name properties and provide information about them
   - (would be slightly inaccurate to say “define” here)
2. state facts about a domain
   - provide information about individuals
3. reason about ontologies and facts
   - determine consequences of what was built and stated

Classes

- What is a Class?
  - e.g., `person`, `pet`, `old`
  - a collection of individuals (object, things, . . .)
  - a way of describing part of the world
  - an object in the world (OWL Full)

Example Classes

```
Class(pp:animal partial
  restriction(pp:eats someValuesFrom(owl:Thing)))
Class(pp:person partial pp:animal)
Class(pp:man complete
  intersectionOf(pp:person pp:male pp:adult))
Class(pp:animal+lover complete
  intersectionOf(pp:person
          restriction(pp:has_pet minCardinality(3))))
```

Example Classes

```
Class(pp:vegetarian complete
  intersectionOf(pp:animal
    restriction(pp:eats
                allValuesFrom(complementOf(pp:animal)))
    restriction(pp:eats
      allValuesFrom(
        complementOf(restriction(pp:part_of
                          someValuesFrom(pp:animal)))))))
DisjointClasses(pp:young pp:adult)
```

Properties

- What is a Property?
  - e.g., `has_father`, `has_pet`, `service_number`
  - a collection of relationships between individuals (and data)
  - a way of describing a kind of relationship between individuals
  - an object in the world (OWL Full)

Example Properties

```
ObjectProperty(pp:eaten_by)
ObjectProperty(pp:eats inverseOf(pp:eaten_by)
                       domain(pp:animal))
ObjectProperty(pp:has_pet domain(pp:person)
                          range(pp:animal))
ObjectProperty(pp:is_pet_of inverseOf(pp:has_pet))
DataProperty(pp:service_number range(xsd:integer))

SubPropertyOf(pp:has_pet pp:likes)
```

Individuals

- objects in the world
- belong to classes
- are related to other objects and to data values via properties

Example Individuals

```
Individual(pp:Tom type(owl:Thing))
Individual(pp:Dewey type(pp:duck))
Individual(pp:Rex type(pp:dog) value(pp:is_pet_of pp:Mick))
Individual(pp:Mick type(pp:male)
    value(pp:reads pp:Daily+Mirror)
    value(pp:drives pp:Q123+ABC))
Individual(pp:The42 type(pp:bus)
    value(pp:service_number "42"^^xsd:integer))
```

Description Logics

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

DL-Lite$_F$

Ontology language:

• Concept inclusion assertions: Cl ⊑ Cr, with:
Cl → A | ∃R | Cl 1 ⊓ Cl 2
Cr → A | ∃R | ¬A | ¬ ∃ R
R → P | P−

• Functionality assertions: (funct R)
Database facts: A(c), P(c, d), with c, d constants

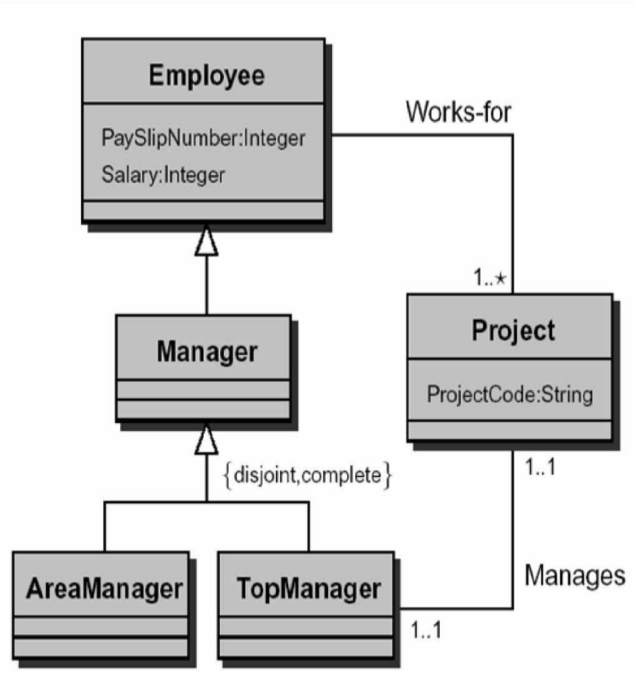
Observations:
• Captures all the basic constructs of ER and UML Class Diagrams

• Notable exception: covering constraints in generalizations

Capturing basic ontology constructs in *DL-Lite*

- ISA between classes → A1 ⊑ A2
- disjointness between classes → A1 ⊑ ¬A2
- domain and range of relations → ∃P ⊑ A1 ∃P$^-$⊑ A2
- mandatory participation → A1 ⊑ ∃P A2 ⊑ ∃P$^-$
- functionality of relations (in *DL-Lite*$_F$) → *(funct P) (funct P)*
- ISA between relations (in *DL-Lite*$_R$) → R1 ⊑ R2

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*

$DL\text{-}Lite_R$

Ontology language:

• Concept inclusion assertions: Cl ⊑ Cr, with:
Cl → A | ∃R | Cl 1 ⊓ Cl 2
Cr → A | ∃R | ¬A | ¬ ∃ R | ∃R.A
R → P | P−

• Role inclusion assertions: R1 ⊑ 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)

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 δ(U_C) (resp. δ(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 ρ*(U_C) (resp. ρ(U_R)), the set of values that U_C(resp. U_R) relates to objects (resp.* pairs of objects). Notice that the domain δ*(U_C) of a concept attribute U_C is a concept,* whereas the domain δ*(U_R) of a role attribute U_R is a role. Furthermore, we denote* with δ_F *(U_C) (resp. δ_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.

– **Concept expressions:**

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

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

– **Value-domain expressions** (*rdfDataType denotes predefined value-domains such* as integers, strings, etc.):

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

$F ::= {>}D \mid E \mid \neg E \mid rdfDataType$

– **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 ⊆ C concept inclusion assertion
- Q ⊆ R role inclusion assertion
- E ⊆ 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.

Ontologia del IMDb

Relativamente allo schema ER presentato precedentemente, viene riportata di seguito l'ontologia completa.

CONCEPTS

ACTOR $\subseteq \exists$ actor2movie

ACTOR $\subseteq \delta$(id)

ACTOR $\subseteq \delta$(name)

ACTOR $\subseteq \delta$(surname)

ACTOR $\subseteq \delta$(sex)

AKANAME $\subseteq \exists$ actorsaka$^-$

AKANAME $\subseteq \delta$(akaname)

AKANAME $\subseteq \delta$(akasurname)

PRODDESIGNER $\subseteq \exists$ proddesignby$^-$

PRODDESIGNER $\subseteq \delta$(name)

PRODDESIGNER $\subseteq \delta$(surname)

DIRECTOR $\subseteq \exists$ directedby$^-$

DIRECTOR $\subseteq \delta$(name)

DIRECTOR $\subseteq \delta$(surname)

PRODUCER $\subseteq \exists$ prodby$^-$

PRODUCER $\subseteq \delta$(name)

PRODUCER $\subseteq \delta$(surname)

AKATITLE $\subseteq \exists$ moviesaka$^-$

AKATITLE $\subseteq \delta$(title)

ITAKATITLE $\subseteq \exists$ moviesitaka$^-$

ITAKATITLE $\subseteq \delta$(title)

CINEMATOGRAPHER $\subseteq \exists$ cinematographyby$^-$

CINEMATOGRAPHER $\subseteq \delta$(name)

CINEMATOGRAPHER $\subseteq \delta$(surname)

DESIGNER $\subseteq \exists$ designer2movie$^-$

DESIGNER $\subseteq \delta$(name)

DESIGNER $\subseteq \delta$(surname)

DISTRIBUTOR $\subseteq \exists$ distributed$^-$

DISTRIBUTOR $\subseteq \delta$(name)

KEYWORD $\subseteq \delta$(word)

PLOT $\subseteq \exists$ plot2movie

PLOT $\subseteq \delta$(id)

PLOT $\subseteq \delta$(plot)

PLOT $\subseteq \delta$(author)

GOOF $\subseteq \exists$ goof2movie

GOOF $\subseteq \delta$(id)

GOOF $\subseteq \delta$(goof)

ALTVERSION $\subseteq \exists$ altversion2movie

ALTVERSION $\subseteq \delta$(id)

ALTVERSION $\subseteq \delta$(version)

TRIVIA $\subseteq \exists$ trivia2movies

TRIVIA $\subseteq \delta$(id)

TRIVIA $\subseteq \delta$(trivia)

TAG $\subseteq \exists$ tag2movie

TAG $\subseteq \delta$(id)

TAG $\subseteq \delta$(tag)

CRAZYCREDIT $\subseteq \exists$ crazycredit2movie

CRAZYCREDIT $\subseteq \delta$(id)

CRAZYCREDIT $\subseteq \delta$(credit)

LANGUAGE $\subseteq \exists$ lang2movie

LANGUAGE $\subseteq \delta$(name)

LINK $\subseteq \exists$ movielinks$^-$

LINK $\subseteq \delta$(id)

LINK $\subseteq \delta$(featured_in)

LINK $\subseteq \delta$(referenced_in)

LINK $\subseteq \delta$(spin)

LINK $\subseteq \delta$(spoofed)

LINK $\subseteq \delta$(version)

LINK $\subseteq \delta$(followed)

LINK $\subseteq \delta$(features)

LINK $\subseteq \delta$(remake)

LINK $\subseteq \delta$(ref)

COUNTRY $\subseteq \exists$ in

COUNTRY $\subseteq \delta$(name)

LOCATION $\subseteq \exists$ in$^-$

LOCATION $\subseteq \delta$(city)

SFXCOMPANY $\subseteq \exists$ country2sfx$^-$

SFXCOMPANY $\subseteq \exists$ sfxby$^-$

SFXCOMPANY $\subseteq \delta$(name)

COMPOSER $\subseteq \exists$ musicby$^-$

COMPOSER $\subseteq \delta$(name)

COMPOSER $\subseteq \delta$(surname)

EDITOR ⊆ ∃ editby⁻

EDITOR ⊆ δ(name)

EDITOR ⊆ δ(surname)

WRITER ⊆ ∃ writerby⁻

WRITER ⊆ δ(name)

WRITER ⊆ δ(surname)

SOUNDTRACK ⊆ δ(name)

SOUNDTRACK ⊆ δ(writer)

SOUNDTRACK ⊆ δ(composer)

SOUNDTRACK ⊆ δ(musician)

PRODCOMPANY ⊆ ∃ prodcompany2movie

PRODCOMPANY ⊆ ∃ prodcompany2country

PRODCOMPANY ⊆ δ(name)

MOVIES ⊆ ∃ lang2movie⁻

MOVIES ⊆ ∃ designer2movie⁻

MOVIES ⊆ ∃ distributed

MOVIES ⊆ ∃ releasein

MOVIES ⊆ ∃ shotin

MOVIES ⊆ ∃ located

MOVIES ⊆ ∃ musicby

MOVIES ⊆ ∃ editby

MOVIES ⊆ ∃ writerby

MOVIES ⊆ ∃ prodcompany2movie⁻

MOVIES ⊆ ∃ actor2movie⁻

MOVIES ⊆ ∃ proddesignby

MOVIES ⊆ ∃ directedby

MOVIES ⊆ ∃ cinematographyby

MOVIES ⊆ δ(id)

MOVIES ⊆ δ(title)

MOVIES ⊆ δ(year)

MOVIES ⊆ δ(runtimes)

MOVIES ⊆ δ(sound)

MOVIES ⊆ δ(genre)

MOVIES ⊆ δ(colortype)

MOVIES ⊆ δ(crewcoverage)

MOVIES ⊆ δ(castcoverage)

MOVIES ⊆ δ(votes)

MOVIES ⊆ δ(rank)

MOVIES ⊆ δ(mpaa)

CERTIFICATE ⊆ ∃ released

CERTIFICATE ⊆ ∃ cert2movie

CERTIFICATE ⊆ δ(type)

ROLES

∃ key2movie^{-} ⊆ MOVIES

∃ key2movie ⊆ KEYWORD

∃ plot2movie^{-} ⊆ MOVIES

∃ plot2movie ⊆ PLOT

∃ goof2movie^{-} ⊆ MOVIES

∃ goof2movie ⊆ GOOF

∃ altversion2movie^{-} ⊆ MOVIES

∃ alversion2movie ⊆ ALTVERSION

∃ trivia2movie⁻ ⊆ MOVIES

∃ trivia2movie ⊆ TRIVIA

∃ tag2movie⁻ ⊆ MOVIES

∃ tag2movie ⊆ TAG

∃ crazycredit2movie⁻ ⊆ MOVIES

∃ crazycredit2movie ⊆ CRAZYCREDIT

∃ lang2movie⁻ ⊆ MOVIES

∃ lang2movie ⊆ LANGUAGE

∃ movielinks⁻ ⊆ LINK

∃ movielinks ⊆ MOVIES

∃ moviesaka⁻ ⊆ AKATITLE

∃ moviesaka ⊆ MOVIES

∃ moviesitaka⁻ ⊆ ITAKATITLE

∃ moviesitaka ⊆ MOVIES

∃ cinematographyby⁻ ⊆ CINEMATOGRAPHER

∃ cinematographyby ⊆ MOVIES

∃ designer2movie⁻ ⊆ DESIGNER

∃ designer2movie ⊆ MOVIES

∃ distributed⁻ ⊆ DISTRIBUTOR

∃ distributed ⊆ MOVIES

∃ releasein⁻ ⊆ COUNTRY

∃ releasein ⊆ MOVIES

∃ located⁻ ⊆ COUNTRY

∃ located ⊆ MOVIES

∃ sfxby⁻ ⊆ SFXCOMPANY

∃ sfxby ⊆ MOVIES

∃ musicby⁻ ⊆ COMPOSER

∃ musicby ⊆ MOVIES

∃ editby⁻ ⊆ EDITOR

∃ editby ⊆ MOVIES

∃ writerby⁻ ⊆ WRITER

∃ writerby ⊆ MOVIES

∃ tracklist⁻ ⊆ SOUNDTRACK

∃ tracklist ⊆ MOVIES

∃ prodcompany2movie⁻ ⊆ PRODCOMPANY

∃ prodcompany2movie ⊆ MOVIES

∃ actor2movie⁻ ⊆ MOVIES

∃ actor2movie ⊆ ACTOR

∃ actorsaka⁻ ⊆ AKANAME

∃ actorsaka ⊆ ACTOR

∃ proddesignby⁻ ⊆ PRODDESIGNER

∃ proddesignby ⊆ MOVIES

∃ directedby⁻ ⊆ DIRECTOR

∃ directedby ⊆ MOVIES

∃ prodby⁻ ⊆ PRODUCER

∃ prodby ⊆ MOVIES

∃ prodcompany2country⁻ ⊆ COUNTRY

∃ prodcompany2country ⊆ PRODCOMPANY

∃ country2sfx⁻ ⊆ SFXCOMPANY

∃ country2sfx ⊆ COUNTRY

∃ in^{-} ⊆ LOCATION

∃ in ⊆ COUNTRY

∃ $distributor2country^{-}$ ⊆ COUNTRY

∃ distributor2country ⊆ DISTRIBUTOR

∃ $released^{-}$ ⊆ COUNTRY

∃ released ⊆ CERTIFICATE

∃ $cert2movie^{-}$ ⊆ MOVIE

∃ cert2movie ⊆ CERTIFICATE

CONCEPTS ATTRIBUTES

ρ(id) ⊆ xsd:int

ρ(name) ⊆ xsd:string

ρ(surname) ⊆ xsd:string

ρ(sex) ⊆ xsd:string

ρ(akaname) ⊆ xsd:string

ρ(akasurname) ⊆ xsd:string

ρ(title) ⊆ xsd:string

ρ(version) ⊆ xsd:string

ρ(type) ⊆ xsd:string

ρ(credit) ⊆ xsd:string

ρ(goof) ⊆ xsd:string

ρ(word) ⊆ xsd:string

ρ(featured_in) ⊆ xsd:string

ρ(referenced_in) ⊆ xsd:string

ρ(spin) ⊆ xsd:string

ρ(spoofed) ⊆ xsd:string

ρ(followed) ⊆ xsd:string

ρ(features) ⊆ xsd:string

ρ(remake) ⊆ xsd:string

ρ(follows) ⊆ xsd:string

ρ(ref) ⊆ xsd:string

ρ(city) ⊆ xsd:string

ρ(year) ⊆ xsd:string

ρ(runtimes) ⊆ xsd:string

ρ(sound) ⊆ xsd:string

ρ(genre) ⊆ xsd:string

ρ(crewcoverage) ⊆ xsd:string

ρ(castcoverage) ⊆ xsd:string

ρ(colortype) ⊆ xsd:string

ρ(votes) ⊆ xsd:int

ρ(rank) ⊆ xsd:float

ρ(mpaa) ⊆ xsd:string

ρ(akatitle) ⊆ xsd:string

ρ(plot) ⊆ xsd:string

ρ(author) ⊆ xsd:string

ρ(writer) ⊆ xsd:string

ρ(composer) ⊆ xsd:string

ρ(musician) ⊆ xsd:string

ρ(tag) ⊆ xsd:string

ρ(trivia) ⊆ xsd:string

ρ(type) ⊆ xsd:string

ROLES ATTRIBUTES

ρ(year) ⊆ xsd:string

ρ(date) ⊆ xsd:string

ρ(role) ⊆ xsd:string

VALUES DOMAIN

ACTOR ⊆ δ(id)

ACTOR ⊆ δ(name)

ACTOR ⊆ δ(surname)

ACTOR ⊆ δ(sex)

AKANAME ⊆ δ(akaname)

AKANAME ⊆ δ(akasurname)

AKATITLE ⊆ δ(title)

ALTVERSION ⊆ δ(id)

ALTVERSION ⊆ δ(version)

CERTIFICATE ⊆ δ(type)

CINEMATOGRAPHER ⊆ δ(name)

CINEMATOGRAPHER ⊆ δ(surname)

COMPOSER ⊆ δ(name)

COMPOSER ⊆ δ(surname)

COUNTRY ⊆ δ(name)

CRAZYCREDIT ⊆ δ(id)

CRAZYCREDIT ⊆ δ(credit)

DESIGNER ⊆ δ(name)

DESIGNER ⊆ δ(surname)

DIRECTOR ⊆ δ(name)

DIRECTOR ⊆ δ(surname)

DISTRIBUTOR ⊆ δ(name)

EDITOR ⊆ δ(name)

EDITOR ⊆ δ(surname)

GOOF ⊆ δ(id)

GOOF ⊆ δ(goof)

ITAKATITLE ⊆ δ(title)

CINEMATOGRAPHER ⊆ δ(name)

CINEMATOGRAPHER ⊆ δ(surname)

KEYWORD ⊆ δ(word)

LANGUAGE ⊆ δ(name)

LINK ⊆ δ(id)

LINK ⊆ δ(featured_in)

LINK ⊆ δ(referenced_in)

LINK ⊆ δ(spin)

LINK ⊆ δ(spoofed)

LINK ⊆ δ(version)

LINK ⊆ δ(followed)

LINK ⊆ δ(features)

LINK ⊆ δ(remake)

LINK ⊆ δ(follows)

LINK ⊆ δ(ref)

LOCATION ⊆ δ(city)

MOVIES ⊆ δ(id)

MOVIES ⊆ δ(title)

MOVIES ⊆ δ(year)

MOVIES ⊆ δ(runtimes)

MOVIES ⊆ δ(sound)

MOVIES ⊆ δ(genre)

MOVIES ⊆ δ(colortype)

MOVIES ⊆ δ(castcoverage)

MOVIES ⊆ δ(crewcoverage)

MOVIES ⊆ δ(votes)

MOVIES ⊆ δ(rank)

MOVIES ⊆ δ(mpaa)

PLOT ⊆ δ(id)

PLOT ⊆ δ(plot)

PLOT ⊆ δ(author)

PRODCOMPANY ⊆ δ(name)

PRODDESIGNER ⊆ δ(name)

PRODDESIGNER ⊆ δ(surname)

PRODUCER ⊆ δ(name)

PRODUCER ⊆ δ(surname)

SFXCOMPANY ⊆ δ(name)

SOUNDTRACK ⊆ δ(name)

SOUNDTRACK ⊆ δ(writer)

SOUNDTRACK ⊆ δ(composer)

SOUNDTRACK ⊆ δ(musician)

TAG ⊆ δ(id)

TAG ⊆ δ(tag)

TRIVIA ⊆ δ(id)

TRIVIA ⊆ δ(trivia)

WRITER ⊆ δ(name)

WRITER ⊆ δ(surname)

shotin ⊆ δ(year)

releasein $\subseteq \delta$(date)

actor2movie $\subseteq \delta$(role)

ATTRIBUTE FUNCTIONALITY

(funct id)

(funct name)

(funct surname)

(funct sex)

(funct akaname)

(funct akasurname)

(funct title)

(funct version)

(funct type)

(funct credit)

(funct goof)

(funct word)

(funct featured_in)

(funct referenced_in)

(funct spin)

(funct spoofed)

(funct followed)

(funct features)

(funct remake)

(funct follows)

(funct ref)

(funct city)

(funct year)

(funct runtimes)

(funct sound)

(funct genre)

(funct crewcoverage)

(funct castcoverage)

(funct colortype)

(funct votes)

(funct rank)

(funct mpaa)

(funct akatitle)

(funct plot)

(funct author)

(funct writer)

(funct composer)

(funct musician)

(funct tag)

(funct trivia)

(funct year)

(funct date)

(funct role)

ROLE FUNCTIONALITY

(funct altversion2movie)

(funct cinematographyby)

(funct crazycredit2movie)

(funct directedby)

(funct distributor2country)

(funct editby)

(funct goof2movie)

(funct plot2movie)

(funct prodcompany2country)

(funct proddesignby)

(funct tag2movie)

(funct trivia2movie)

(funct actorsaka⁻)

(funct moviesaka⁻)

(funct moviesitaka⁻)

(funct movielinks⁻)

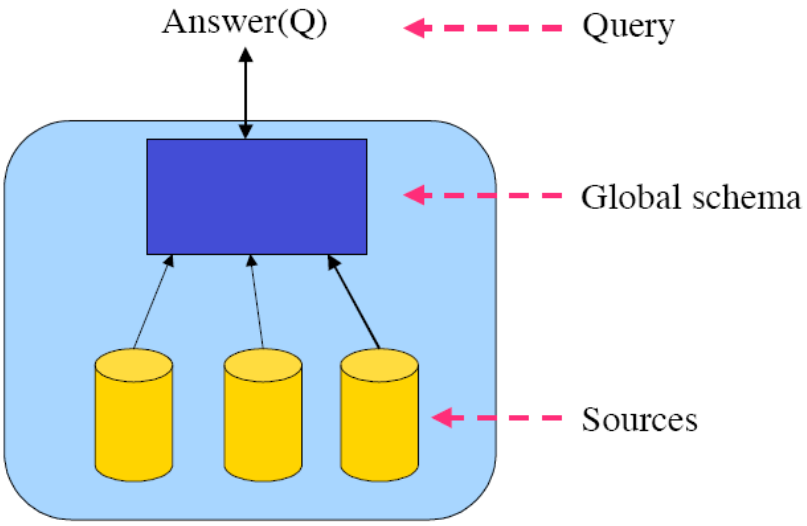
(funct in⁻)

(funct released)

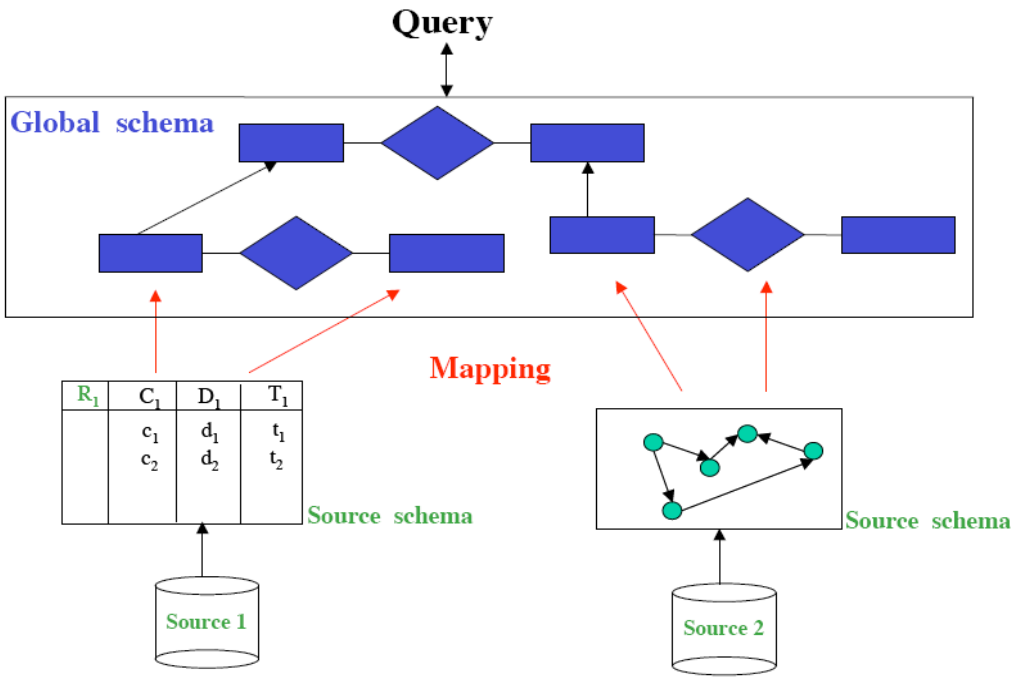
(funct cert2movie)

Mapping

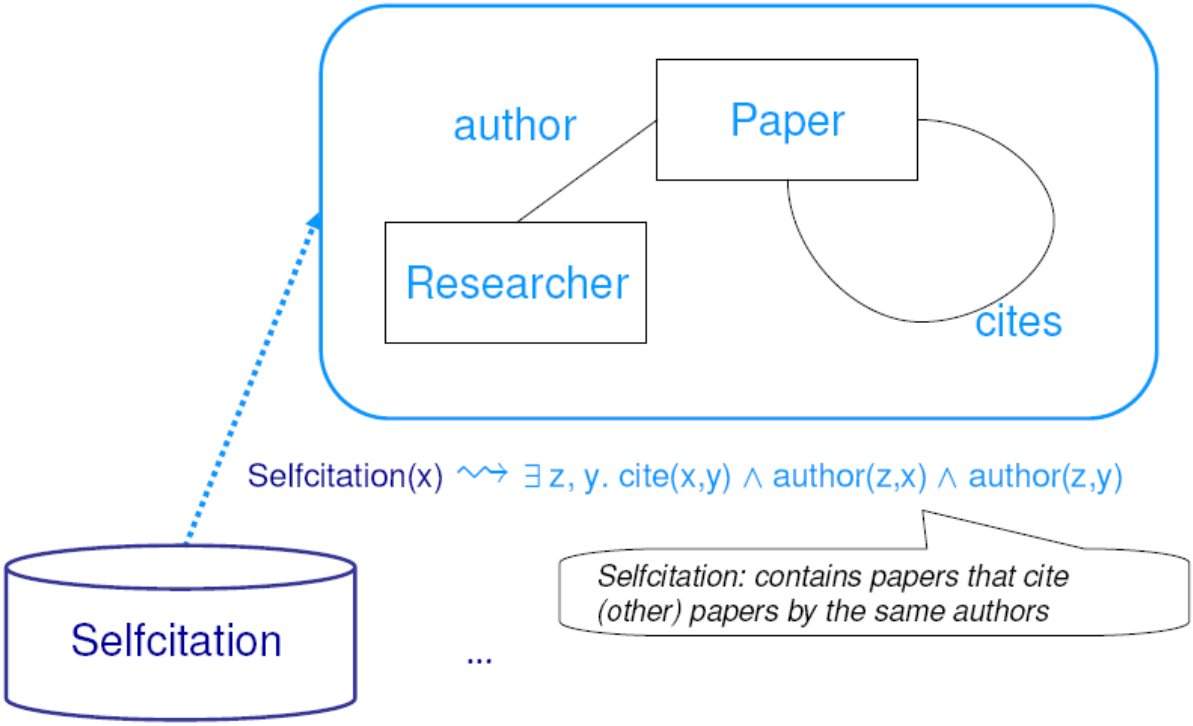
Data integration



Data integration



An example



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 : an example to understand

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 $\forall \vec{\mathbf{x}} \, (s(\vec{\mathbf{x}}) \rightarrow \phi_{\mathcal{G}}(\vec{\mathbf{x}}))$.

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

$$\mathsf{r}_1(T, Y, D) \rightsquigarrow \{ (T, Y, D) \mid \mathsf{movie}(T, Y, D) \wedge \mathsf{european}(D) \wedge Y \geq 1960 \}$$
$$\mathsf{r}_2(T, R) \rightsquigarrow \{ (T, R) \mid \mathsf{movie}(T, Y, D) \wedge \mathsf{review}(T, R) \wedge Y \geq 1990 \}$$

The query $\{ (T, R) \mid \mathsf{movie}(T, 1998, D) \wedge \mathsf{review}(T, R) \}$ 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

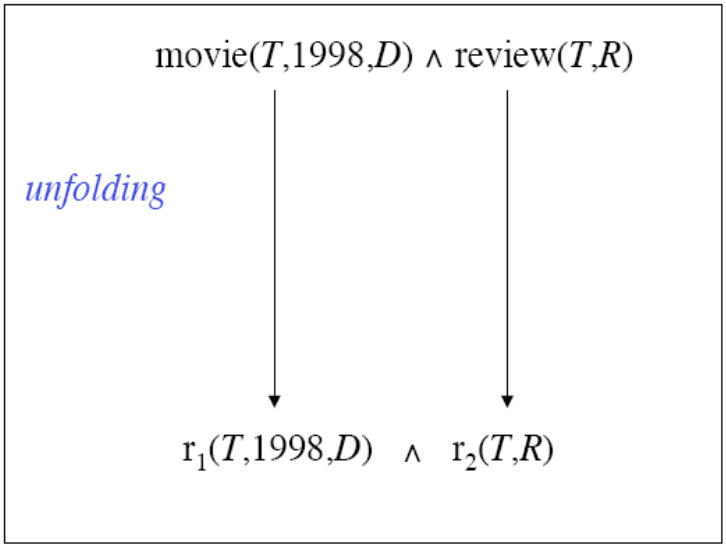
movie$(T, Y, D) \rightsquigarrow \{ (T, Y, D) \mid \mathsf{r}_1(T, Y, D) \}$

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

Di seguito vengono presentati i mapping tra l'ontologia costruita in DL-Lite e il database relazionale.

Ma chiariamo meglio il concetto di 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 ma maggiori s pping 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

$$\phi \rightarrow T_i$$

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

$$\phi \rightarrow \psi$$

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.

IMDb: typing mapping assertion

M_{t1} : SELECT id FROM ACTOR ⇝ xsd:int

M_{t2} : SELECT name FROM ACTOR ⇝ xsd:string

M_{t3}: SELECT surname FROM ACTOR ⇝ xsd:string

M_{t4}: SELECT sex FROM ACTOR ⇝ xsd:string

M_{t5}: SELECT actor FROM actor2movie ⇝ xsd:int

M_{t6} : SELECT movie FROM actor2movie ⇝ xsd:int

M_{t7} : SELECT role FROM actor2movie ⇝ xsd:string

M_{t8} : SELECT actor FROM actorsaka ⇝ xsd:int

M_{t9}: SELECT akasurname FROM actorsaka ⇝ xsd:string

M_{t10}: SELECT akasurname FROM actorsaka ⇝ xsd:string

M_{t11}: SELECT akaname FROM AKANAME ⇝ xsd:string

M_{t12} : SELECT akasurname FROM AKANAME ⇝ xsd:string

M_{t13}: SELECT title FROM AKATITLE ⇝ xsd:string

M_{t14}: SELECT id FROM ALTVERSION ⇝ xsd:int

M_{t15}: SELECT version FROM ALTVERSION ⇝ xsd:string

M_{t16}: SELECT altversion FROM altversion2movie ⇝ xsd:int

M_{t17}: SELECT movie FROM altversion2movie ⇝ xsd:int

M_{t18}: SELECT country FROM CERTIFICATE ⇝ xsd:string

M_{t19}: SELECT movie FROM CERTIFICATE ⇝ xsd:int

M_{t20}: SELECT type FROM CERTIFICATE ⇝ xsd:string

M_{t21}: SELECT name FROM CINEMATOGRAPHER ⇝ xsd:string

M_{t22}: SELECT surname FROM CINEMATOGRAPHER ⇝ xsd:string

$\mathbf{M_{t23}}$: SELECT movie FROM cinematographyby ⇝ xsd:int

$\mathbf{M_{t24}}$: SELECT cinematographerN FROM cinematographyby ⇝ xsd:string

$\mathbf{M_{t25}}$: SELECT cinematographerS FROM cinematographyby ⇝ xsd:string

$\mathbf{M_{t26}}$: SELECT name FROM COMPOSER ⇝ xsd:string

$\mathbf{M_{t27}}$: SELECT surname FROM COMPOSER ⇝ xsd:string

$\mathbf{M_{t28}}$: SELECT name FROM COUNTRY ⇝ xsd:string

$\mathbf{M_{t29}}$: SELECT sfxcompany FROM country2sfx ⇝ xsd:string

$\mathbf{M_{t30}}$: SELECT country FROM country2sfx ⇝ xsd:string

$\mathbf{M_{t31}}$: SELECT id FROM CRAZYCREDIT ⇝ xsd:int

$\mathbf{M_{t32}}$: SELECT credit FROM CRAZYCREDIT ⇝ xsd:string

$\mathbf{M_{t33}}$: SELECT credit FROM crazycredit2movie ⇝ xsd:int

$\mathbf{M_{t34}}$: SELECT movie FROM crazycredit2movie ⇝ xsd:int

$\mathbf{M_{t35}}$: SELECT name FROM DESIGNER ⇝ xsd:string

$\mathbf{M_{t36}}$: SELECT surname FROM DESIGNER ⇝ xsd:string

$\mathbf{M_{t37}}$: SELECT movie FROM designer2movie ⇝ xsd:int

$\mathbf{M_{t38}}$: SELECT designerN FROM designer2movie ⇝ xsd:string

$\mathbf{M_{t39}}$: SELECT designerS FROM designer2movie ⇝ xsd:string

$\mathbf{M_{t40}}$: SELECT name FROM DIRECTOR ⇝ xsd:string

$\mathbf{M_{t41}}$: SELECT surname FROM DIRECTOR ⇝ xsd:string

$\mathbf{M_{t42}}$: SELECT movie FROM directedby ⇝ xsd:int

$\mathbf{M_{t43}}$: SELECT directorN FROM directedby ⇝ xsd:string

$\mathbf{M_{t44}}$: SELECT directorS FROM directedby ⇝ xsd:string

$\mathbf{M_{t45}}$: SELECT distributor FROM distributed ⇝ xsd:string

$\mathbf{M_{t46}}$: SELECT movie FROM distributed ⇝ xsd:int

$\mathbf{M_{t47}}$: SELECT name FROM DISTRIBUTOR ⇝ xsd:string

$\mathbf{M_{t48}}$: SELECT distributor FROM distributor2country ⇝ xsd:string

$\mathbf{M_{t49}}$: SELECT country FROM distributor2country ⇝ xsd:string

$\mathbf{M_{t50}}$: SELECT name FROM EDITOR ⇝ xsd:string

$\mathbf{M_{t51}}$: SELECT surname FROM EDITOR ⇝ xsd:string

$\mathbf{M_{t52}}$: SELECT movie FROM editby ⇝ xsd:int

$\mathbf{M_{t53}}$: SELECT editorN FROM editby ⇝ xsd:string

$\mathbf{M_{t54}}$: SELECT editorS FROM editby ⇝ xsd:string

$\mathbf{M_{t55}}$: SELECT id FROM GOOF ⇝ xsd:int

$\mathbf{M_{t56}}$: SELECT goof FROM GOOF ⇝ xsd:string

$\mathbf{M_{t57}}$: SELECT goof FROM goof2movie ⇝ xsd:int

$\mathbf{M_{t58}}$: SELECT movie FROM goof2movie ⇝ xsd:int

$\mathbf{M_{t59}}$: SELECT title FROM ITAKATITLE ⇝ xsd:string

$\mathbf{M_{t60}}$: SELECT word FROM KEYWORD ⇝ xsd:string

$\mathbf{M_{t61}}$: SELECT keyword FROM key2movie ⇝ xsd:string

$\mathbf{M_{t62}}$: SELECT movie FROM key2movie ⇝ xsd:int

$\mathbf{M_{t63}}$: SELECT name FROM LANGUAGE ⇝ xsd:string

$\mathbf{M_{t64}}$: SELECT language FROM lang2movie ⇝ xsd:string

$\mathbf{M_{t65}}$: SELECT movie FROM lang2movie ⇝ xsd:int

$\mathbf{M_{t66}}$: SELECT id FROM LINK ⇝ xsd:int

$\mathbf{M_{t67}}$: SELECT featured_in FROM LINK ⇝ xsd:string

$\mathbf{M_{t68}}$: SELECT referenced_in FROM LINK ⇝ xsd:string

$\mathbf{M_{t69}}$: SELECT spin FROM LINK ⇝ xsd:string

$\mathbf{M_{t70}}$: SELECT spoofed FROM LINK ⇝ xsd:string

$\mathbf{M_{t71}}$: SELECT version FROM LINK ⇝ xsd:string

$\mathbf{M_{t72}}$: SELECT followed FROM LINK ⇝ xsd:string

$\mathbf{M_{t73}}$: SELECT features FROM LINK ⇝ xsd:string

$\mathbf{M_{t74}}$: SELECT remake FROM LINK ⇝ xsd:string

$\mathbf{M_{t75}}$: SELECT follows FROM LINK ⇝ xsd:string

$\mathbf{M_{t76}}$: SELECT ref FROM LINK ⇝ xsd:string

$\mathbf{M_{t77}}$: SELECT locationcity FROM located ⇝ xsd:string

$\mathbf{M_{t78}}$: SELECT locationcountry FROM located ⇝ xsd:string

$\mathbf{M_{t79}}$: SELECT movie FROM located ⇝ xsd:int

$\mathbf{M_{t80}}$: SELECT city FROM LOCATION ⇝ xsd:string

$\mathbf{M_{t81}}$: SELECT country FROM LOCATION ⇝ xsd:string

$\mathbf{M_{t82}}$: SELECT link FROM movielinks ⇝ xsd:int

$\mathbf{M_{t83}}$: SELECT movie FROM movielinks ⇝ xsd:int

$\mathbf{M_{t84}}$: SELECT id FROM MOVIES ⇝ xsd:int

$\mathbf{M_{t85}}$: SELECT title FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t86}}$: SELECT year FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t87}}$: SELECT runtimes FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t88}}$: SELECT sound FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t89}}$: SELECT genre FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t90}}$: SELECT colortype FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t91}}$: SELECT crewcoverage FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t92}}$: SELECT castcoverage FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t93}}$: SELECT votes FROM MOVIES ⇝ xsd:int

$\mathbf{M_{t94}}$: SELECT rank FROM MOVIES ⇝ xsd:float

$\mathbf{M_{t95}}$: SELECT mpaa FROM MOVIES ⇝ xsd:string

$\mathbf{M_{t96}}$: SELECT akatitle FROM moviesaka ⇝ xsd:string

$\mathbf{M_{t97}}$: SELECT movie FROM moviesaka ⇝ xsd:int

$\mathbf{M_{t98}}$: SELECT itakatitle FROM moviesitaka ⇝ xsd:string

$\mathbf{M_{t99}}$: SELECT movie FROM moviesitaka ⇝ xsd:int

$\mathbf{M_{t100}}$: SELECT movie FROM musicby ⇝ xsd:int

$\mathbf{M_{t101}}$: SELECT composerN FROM musicby ⇝ xsd:string

$\mathbf{M_{t102}}$: SELECT composerS FROM musicby ⇝ xsd:string

$\mathbf{M_{t103}}$: SELECT id FROM PLOT ⇝ xsd:int

$\mathbf{M_{t104}}$: SELECT plot FROM PLOT ⇝ xsd:string

$\mathbf{M_{t105}}$: SELECT author FROM PLOT ⇝ xsd:string

$\mathbf{M_{t106}}$: SELECT plot FROM plot2movie ⇝ xsd:int

$\mathbf{M_{t107}}$: SELECT movie FROM plot2movie ⇝ xsd:int

$\mathbf{M_{t108}}$: SELECT movie FROM prodby ⇝ xsd:int

$\mathbf{M_{t109}}$: SELECT producerN FROM prodby ⇝ xsd:string

$\mathbf{M_{t110}}$: SELECT producerS FROM prodby ⇝ xsd:string

$\mathbf{M_{t111}}$: SELECT name FROM PRODCOMPANY ⇝ xsd:string

$\mathbf{M_{t112}}$: SELECT prodcompany FROM prodcompany2country ⇝ xsd:string

$\mathbf{M_{t113}}$: SELECT country FROM prodcompany2country ⇝ xsd:string

$\mathbf{M_{t114}}$: SELECT prodcompany FROM prodcompany2movie ⇝ xsd:string

$\mathbf{M_{t115}}$: SELECT movie FROM prodcompany2movie ⇝ xsd:string

$\mathbf{M_{t116}}$: SELECT name FROM PRODDESIGNER ⇝ xsd:string

$\mathbf{M_{t117}}$: SELECT surname FROM PRODDESIGNER ⇝ xsd:string

$\mathbf{M_{t118}}$: SELECT movie FROM proddesignby ⇝ xsd:int

$\mathbf{M_{t119}}$: SELECT proddesignerN FROM proddesignby ⇝ xsd:string

$\mathbf{M_{t120}}$: SELECT proddesignerS FROM proddesignby ⇝ xsd:string

$\mathbf{M_{t121}}$: SELECT name FROM PRODUCER ⇝ xsd:string

$\mathbf{M_{t122}}$: SELECT surname FROM PRODUCER ⇝ xsd:string

$\mathbf{M_{t123}}$: SELECT movie FROM releasein ⇝ xsd:int

$\mathbf{M_{t124}}$: SELECT country FROM releasein ⇝ xsd:string

$\mathbf{M_{t125}}$: SELECT date FROM releasein ⇝ xsd:string

$\mathbf{M_{t126}}$: SELECT movie FROM sfxby ⇝ xsd:int

$\mathbf{M_{t127}}$: SELECT sfxcompany FROM sfxby ⇝ xsd:string

$\mathbf{M_{t128}}$: SELECT name FROM SFXCOMPANY ⇝ xsd:string

$\mathbf{M_{t129}}$: SELECT movie FROM shotin ⇝ xsd:int

$\mathbf{M_{t130}}$: SELECT country FROM shotin ⇝ xsd:string

$\mathbf{M_{t131}}$: SELECT year FROM shotin ⇝ xsd:string

$\mathbf{M_{t132}}$: SELECT name FROM SOUNDTRACK ⇝ xsd:string

$\mathbf{M_{t133}}$: SELECT writer FROM SOUNDTRACK ⇝ xsd:string

$\mathbf{M_{t134}}$: SELECT composer FROM SOUNDTRACK ⇝ xsd:string

$\mathbf{M_{t135}}$: SELECT musician FROM SOUNDTRACK ⇝ xsd:string

$\mathbf{M_{t136}}$: SELECT id FROM TAG ⇝ xsd:int

$\mathbf{M_{t137}}$: SELECT tag FROM TAG ⇝ xsd:string

$\mathbf{M_{t138}}$: SELECT tag FROM tag2movie ⇝ xsd:int

$\mathbf{M_{t139}}$: SELECT movie FROM tag2movie ⇝ xsd:int

$\mathbf{M_{t140}}$: SELECT soundtrack FROM SOUNDTRACK ⇝ xsd:string

$\mathbf{M_{t141}}$: SELECT movie FROM SOUNDTRACK ⇝ xsd:int

$\mathbf{M_{t142}}$: SELECT id FROM TRIVIA ⇝ xsd:int

$\mathbf{M_{t143}}$: SELECT trivia FROM TRIVIA ⇝ xsd:string

$\mathbf{M_{t144}}$: SELECT trivia FROM trivia2movies ⇝ xsd:int

$\mathbf{M_{t145}}$: SELECT movie FROM trivia2movies ⇝ xsd:int

$\mathbf{M_{t146}}$: SELECT name FROM WRITER ⇝ xsd:int

$\mathbf{M_{t147}}$: SELECT surname FROM WRITER ⇝ xsd:string

$\mathbf{M_{t148}}$: SELECT movie FROM writerby ⇝ xsd:int

$\mathbf{M_{t149}}$: SELECT writerN FROM writerby ⇝ xsd:string

$\mathbf{M_{t150}}$: SELECT writerS FROM writerby ⇝ xsd:string

IMDb: data-to-object mapping assertions

Λ(OBJECT VARIABLE TERMS)={actor(x),akaname(x,y),akatitle(x),altversion(x),certificate(x),cinematographer(x,y),composer(x,y),country(x),crazycredit(x),designer(x,y),director(x,y),distributor(x),editor(x,y),goof(x),itakatitle(x),keyword(x),language(x),link(x),location(x),movie(x),plot(x),prodcompany(x),proddesigner(x,y),producer(x,y),sfxcompany(x),soundtrack(x),tag(x),trivia(x),writer(x,y)}

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*)

M_{m2}: SELECT *actor,movie,role*
FROM actor2movie

actor2movie(**actor** (*actor*), **movie** (*movie*)), role(**actor** (*actor*), **movie** (*movie*), *role*)

M_{m3}: SELECT *actor,akaname,akasurname*
FROM actorsaka

actorsaka(**actor** (*actor*),**akaname** (*akaname,akasurname*))

M_{m4}: SELECT *akaname,akasurname*
FROM AKANAME

AKANAME(**akaname** (*akaname,akasurname*)), akaname(**akaname** (*akaname,akasurname*), *akaname),* akasurname(**akaname** (*akaname,akasurname*), *akasurname)*

M_{m5}: SELECT *title*
FROM AKATITLE

AKATITLE(**akatitle** (*title*)), title(**akatitle** (*title*), *title*)

M_{m6}: SELECT *id,version*
FROM ALTVERSION

ALTVERSION(**altversion** (*id*)), id(**altversion** (*id*), *id),* version(**altversion** (*id*), *version*)

M_{m7}: SELECT *version,movie*
FROM altversion2movie

altversion2movie(**altversion** (*version*),**movie** (*movie*))

M_{m8}: SELECT *country,movie,type*
FROM CERTIFICATE

CERTIFICATE(**country** (*country),***movie** (*movie*)), type(**certificate** (*type*), *type*)

M_{m9}: SELECT *name,surname*
FROM CINEMATOGRAPHER

CINEMATOGRAPHER(**cinematographer** (*name, surname*)), name(**cinematographer** (*name, surname*), *name),* surname(**cinematographer** (*name, surname*), *surname)*

M_{m10}: SELECT *movie,cinematographerN,cinematographerS*
FROM cinematographyby

cinematographyby(**movie** (*movie*),**cinematographer** (*cinematographerN, cinematographerS*))

M_{m11}: SELECT *name,surname*
FROM COMPOSER

COMPOSER(**composer** (*name, surname*)), name(**composer** (*name, surname*), *name),* surname(**composer** (*name, surname*), *surname)*

M_{m12}: SELECT *name*
FROM COUNTRY

COUNTRY(**country** (*name*)), name(**country** (*name*), *name*)

M_{m13}: SELECT *sfxcompany,country*
FROM country2sfx

country2sfx(**sfxcompany** (*sfxcompany*), **country** (*country*))

M_{m14}: SELECT *id,credit*
FROM CRAZYCREDIT

CRAZYCREDIT(**crazycredit** (*id*)), id(**crazycredit** (*id*), *id),* credit(**crazycredit** (*id*), *credit*)

M_{m15}: SELECT *credit,movie*
FROM crazycredit2movie

crazycredit2movie(**crazycredit**(*credit*),**movie** (*movie*))

M_{m16}: SELECT *name,surname*
FROM DESIGNER

DESIGNER(**designer** (*name, surname*)), name(**designer** (*name, surname*), *name),* surname(**designer** (*name, surname*), *surname)*

M_{m17}: SELECT *designerN,designerS,movie*
FROM designer2movie

designer2movie(**designer** (*designerN,designerS*), **movie** (*movie*))

M_{m18}: SELECT *movie,directorN,directorS*
FROM directedby

directedby(**movie** (*movie*), **director** (*directorN,directorS*))

M_{m19}: SELECT *name,surname*
FROM DIRECTOR

DIRECTOR(**director** (*name, surname*)), name(**director** (*name, surname*), *name),* surname(**director** (*name, surname*), *surname)*

M_{m20}: SELECT *name*
FROM DISTRIBUTOR

DISTRIBUTOR(**distributor** (*name*)), name(**distributor** (*name*), *name*)

M_{m21}: SELECT *distributor,movie*
FROM distributed

distributed(**distributor** (*distributor*), **movie** (*movie*))

M_{m22}: SELECT *distributor,country*
FROM distributor2country

distributor2country (**distributor** (*distributor*), **country** (*country*))

M_{m23}: SELECT *movie,editorN,editorS*
FROM editby

editby(**movie** (*movie*), **editor** (*editorN,editorS*))

M_{m24}: SELECT *name,surname*
FROM EDITOR

EDITOR(**editor** (*name, surname*)), name(**editor** (*name, surname*), *name),* surname(**editor** (*name, surname*), *surname)*

M_{m25}: SELECT *id,goof*
FROM GOOF

GOOF (**goof** (*id*)), id(**goof** (*id*), *id,* goof(**goof** (*id*), *goof*)

M_{m26}: SELECT *goof,movie*
FROM goof2movie

goof2movie(**goof**(*goof*),**movie** (*movie*))

M_{m27}: SELECT *title*
FROM ITAKATITLE

ITAKATITLE(**itakatitle** (*title*)), title(**itakatitle** (*title*), *title*)

M_{m28}: SELECT *word*
FROM KEYWORD

KEYWORD(**keyword** (*word*)), word(**keyword** (*word*), *word*)

M_{m29}: SELECT *keyword,movie*
FROM key2movie

key2movie(**movie** (*movie*),**keyword**(*keyword*))

M_{m30}: SELECT *name*
FROM LANGUAGE

LANGUAGE(**language**(*name*)), name(**language**(*name*), *name*)

M_{m31}: SELECT *movie,language*
FROM lang2movie

lang2movie(**movie** (*movie*),**language**(*language*))

M_{m32}: SELECT
id,featured_in,referenced_in,spin,spoofed,version,followed,features,remake,follows,ref
FROM link

LINK(**link** (*id*)), id(**link** (*id*), *id*), featured_in(**link** (*id*), *featured_in*), referenced_in(**link** (*id*), *referenced_in*), spin(**link** (*id*), *spin*), spoofed(**link** (*id*), *spoofed*), version(**link** (*id*), *version*), followed(**link** (*id*), *followed*), features(**link** (*id*), *features*), remake(**link** (*id*), *remake*), follows(**link** (*id*), *follows*), ref(**link** (*id*), *ref*)

M_{m33}: SELECT *locationcity,locationcountry,movie*
FROM located

located(**country**(*locationcity*),**location**(*locationcountry*), **movie**(*movie*))

M_{m34}: SELECT *link,movie*
FROM movielinks

movielinks(**link** (*link*),**movie**(*movie*))

M_{m35}: SELECT
id,title,year,runtimes,sound,genre,colortype,crewcoverage,castcoverage,votes,rank,mpaa
FROM MOVIES

MOVIES(**movie**(*id*)), id(**movie**(*id*), *id*), title(**movie** (*id*),*title*), year(**movie** (*id*), *year*), runtimes(**movie** (*id*), *runtimes*), sound(**movie** (*id*), *sound*), genre(**movie** (*id*), *genre*), colortype(**movie** (*id*),*colortype*), crewcoverage(**movie** (*id*), *crewcoverage*), castcoverage(**movie** (*id*), *castcoverage*), votes(**movie** (*id*),*votes*), rank(**movie** (*id*), *rank*), mpaa(**movie** (*id*), *mpaa*)

M_{m36}: SELECT *akatitle,movie*
FROM moviesaka

moviesaka(**akatitle** (*akatitle*),**movie**(*movie*))

M_{m37}: SELECT *itakatitle,movie*
FROM moviesitaka

moviesitaka(**itakatitle**(*itakatitle*),**movie**(*movie*))

M_{m38}: SELECT *composerN,composerS,movie*
FROM musicby

musicby(**composer** (*composerN,composerS*),**movie** (*movie*)*)*

M_{m39}: SELECT *id,plot,author*
FROM PLOT

PLOT **plot** (*id)*), id(**plot** (*id),id*), plot*(***plot** (*id*), *plot),* author(**plot** (*id*), *author)*

M_{m40}: SELECT *plot,movie*
FROM plot2movie

plot2movie(**plot**(*plot*), **movie** (*movie*))

M_{m41}: SELECT *producerN,producerS,movie*
FROM prodby

prodby(**producer** (*producerN,producerS*),**movie** (*movie*)*)*

M_{m42}: SELECT *name*
FROM PRODCOMPANY

PRODCOMPANY (**prodcompany**(*name*)), name(**prodcompany**(*name*), *name*)

M_{m43}: SELECT *prodcompany,country*
FROM prodcompany2country

prodcompany2country (**prodcompany** (*prodcompany*), **country** (*country*))

M_{m44}: SELECT *prodcompany,movie*
FROM prodcompany2movie

prodcompany2movie (**prodcompany** (*prodcompany*), **movie** (*movie*))

M_{m45}: SELECT *movie,proddesignerN,proddesignerS*
FROM proddesignby

proddesignby(**movie** (*movie*), **proddesigner** (*proddesignerN,proddesignerS*))

M_{m46}: SELECT *name,surname*
FROM PRODDESIGNER

PRODDESIGNER(**proddesigner** (*name, surname*)), name(**proddesigner** (*name, surname*), *name),* surname(**proddesigner** (*name, surname*), *surname)*

M_{m47}: SELECT *name,surname*
FROM PRODUCER

PRODUCER(**producer** (*name, surname*)), name(**producer** (*name, surname*), *name),* surname(**producer** (*name, surname*), *surname)*

M_{m48}: SELECT *movie,country,date*
FROM releasein

releasein(**movie** (*movie*), **country** (*country*)), date(**movie** (*movie*),**country** (*country*),*date*)

M_{m49}: SELECT *sfxcompany,movie*
FROM sfxby

sfxby(**sfxcompany**(*sfxcompany*), **movie** (*movie*))

M_{m50}: SELECT *country, movie,year*
FROM shotin

shotin(**country** (*country*), **movie** (*movie*), year(**movie** (*movie*),**country** (*country*),*year*))

M_{m51}: SELECT *name,writer,composer,musician*
FROM SOUNDTRACK

SOUNDTRACK(**soundtrack** (*name*)), name(**soundtrack** (*name*), *name),* writer(**soundtrack** (*name*), *writer),* composer(**soundtrack** (*name*), *composer),* musician(**soundtrack** (*name*), *musician)*

M_{m52}: SELECT *id,tag*
FROM TAG

TAG(**tag**(*id*)), id(**tag**(*id*), *id),* tag(**tag** (*id*), *tag*)

M$_{m53}$: SELECT *tag,movie*
FROM tag2movie

tag2movie(**tag** (*tag*),**movie** (*movie*))

M$_{m54}$: SELECT *soundtrack,movie*
FROM tracklist

tracklist(**soundtrack** (*soundtrack*),**movie** (*movie*))

M$_{m55}$: SELECT *id,trivia*
FROM TRIVIA

TRIVIA(**trivia**(*id*)), id(**trivia**(*id*), *id),* trivia(**trivia** (*id*),*trivia*)

M$_{m56}$: SELECT *trivia,movie*
FROM trivia2movie

trivia2movie(**trivia** (*trivia*),**movie** (*movie*))

M$_{m57}$: SELECT *name,surname*
FROM WRITER

WRITER(**writer** (*name, surname*)), name(**writer** (*name, surname*), *name),* surname(**writer**(*name, surname*), *surname)*

M$_{m58}$: SELECT *writerN,writerS*, *movie,*
FROM writerby

writerby(**writer**(*writerN,writerS*), **movie** (*movie*))

M_{m59}: SELECT *city,country*
FROM LOCATION

LOCATION(**location** (*city*)), city(**location** (*city*),*city),* **country**(*country*)

M_{m60}: SELECT *name*
FROM SFXCOMPANY

SFXCOMPANY(**sfxcompany** (*name*)), name(**sfxcompany** (*name*),*name)*

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.