

# Integration Models, Information Integration, CSCI 6967-01

## 1 Motivating Example

Suppose we want to create an application to provide information about movies. The Internet Movie Database (IMDB) contains detailed information about movies including title, actors, year made, release date, etc. It also links to pages containing reviews of this movie. The yahoo box office (YBO) contains listing of the box office performance of the recent movies as well as movies that made a lot of money (top 20 all time grossing movies). The Times Union (TU) website lists the movies playing locally with movie times. Fandango lists the movies for all movie theaters theaters (not only local movie theaters) associated with them. Roger Eberts reviews include a rating and a text, while Washington Post reviews include only text with user reviews. Rotten tomatoes (RT) include multiple ratings, overall, community, top critics, etc. as well as box office success, actors, director, etc. It should be noted though that both IMDB and RT have a search engine that will search for the entered text, but they have more specialized engines where you can search within an era or genre on RT, and you can search within a specific attribute (title, actor, etc.) within IMDB.

Our idea is to be able to support queries that no single source cannot answer “as is” or without the help of information from another source. Some examples are:

- find actors who played in top 20 most grossing movies of all times
- find the movies made in 2007 rated 4 stars by Ebert
- find the Washington post review for the movie “Cloverfield”
- find the movie times for the all the movies with the 80% or higher tomato meter for top reviewers in my area
- find the box office performance of all movies released on July 4, 2006.

How are we going to use the above sources to answer these queries? There are actually many possible ways for some. So, let’s see the possible approaches.

### 1.1 Global as view (GAV)

So, let’s assume that the queries that are going to be asked by a user are known in advance (as would be true for any application). Hence, we have a set of relations (tables) that the user is going to query:

---

*top20Movie(M), playedIn(M, A), movieYear(M, Y), ratedBy(M, Rater),  
ebertRating(M, X, R), waPoReview(M, X, R), rtReview(M, X, Y), rtBoxOffice(M, X),  
yahooBoxOffice(M, X), ...*

---

This constitutes your data model. We will call it the global view, the users will only know the global view and ask queries using this view. This is similar to setting up a database with these relations. However, the relations are virtual, they are not stored on a local database.

Now, we have to map these information stored in the sources to these relations. These tell us how we can populate these relations.

For example, IMDB can provide information about movies. For example, we can find who directed a movie, who starred in a movie, etc if we supply the name of the movie using a faceted search. Note that another program called a wrapper will take your query, formulate it correctly for the specific source using its query language (or use a web service if one is available) and then parse the returned results (either an RDF/XML or pure HTML format) and return a single relation containing only the attributes we are interested in. Basically, we can hide all that complexity, and write simply:

---

*movie(M)@IMDB, title(M, \$X)@IMDB, madeIn(M, Y)@IMDB → movieYear(M, Y)*  
*movie(M)@IMDB, title(M, \$X)@IMDB, reviewedBy(M, Y, Z)@IMDB → reviewLink(Y, Z)*

---

which means that we should ask IMDB that we are interested in a movie with the title equivalent to our given title and find its year (\$ symbol denotes what is an input to this program). This will give us the year of the movie. Note that IMDB contains TV shows also, so we must be specific in our query to find the correct info.

We can now write other relations, such as who acted in a movie and who reviewed this movie etc. Suppose we are querying RT. We have a different way of writing the same query:

---

*contains(M, \$X)@RT, madeIn(M, Y)@RT → movieYear(M, Y)*  
*contains(M, \$X)@RT, criticReview(M, Y)@RT, userReview(M, Z)@RT → rottenReview(M, Y + Z/2)*  
*contains(M, \$X)@RT, boxOffice(M, Y), Y > 0 → boxOffice(M, Y)*

---

Now, contains and title actually refer to methods that we will invoke to compute this relation. For *contains*, we will have to retrieve the results first and then parse them to find the best match for our title. This is just the application complexity being exposed to the application programmer.

Yahoo box office:

---

*top20(\$M)@YBO → top20Movie(M)*  
*boxOffice(\$M, Y)@YBO → boxOffice(M, Y)*

---

Chiago Sun-Times:

---

$$reviewLink(\$Y, ebert)@CST, review(\$Y, Z, W) \rightarrow ebertReview(Y, Z, W)$$

---

Basically, each is a view definition over the relations stored in each relation and we define the global relations as views over the local relations.

The user is now given these virtual relations against which they can ask their queries. A user query is then given by a conjunction of the global predicates. Examples:

Find actors who played in top 20 most grossing movies of all times:

---

$$answer(A) : \neg top20Movie(M), playedIn(M, A)$$

---

Find the movies made in 2007 rated 4 stars by Ebert:

---

$$answer(M) : \neg movieYear(M, 2007), ebertReview(M, 4, Z)$$

---

We can now choose a method to obtain each one and execute. We must execute all possible ways to find the complete answer to this query.

Unfortunately, the left hand side is treated more or less as a way to answer this query. It does not contain more detailed information the contents of the source. How can we say that yahoo box office is only for recent movies? How can we say that timesunion only supports the local movies and fandango supports the movies for fandango affiliates? This requires information that is not necessarily stored in that database. For example, the Yahoo database only stores information about the movie names, not their release dates. Furthermore, we have been treating the right hand side of the relations as methods to obtain the relation from the base relations the source contains, not as a way to define what they contain with respect to the global view.

## 1.2 Local as View (LAV)

Now, suppose we do not care how the results are obtained from the sources (though eventually we will care). We will still have a global set of predicates against which the users will ask their views using conjunctions and other possible query constructs. Now, let's describe the contents of the information sources with respect to these predicates.

Now, we can use a more comprehensive series of relations to describe the information contained in the sources.

IMDB:

---

$$IMDB(M, Y, R) : \neg movie(M), movieYear(M, Y), releaseDate(M, R)$$

---

Yahoo Box Office:

---

$$YBO1(M, Y) : \neg movie(M), releaseDate(M, R), R > '01/28/2008', boxOffice(M, Y)$$
$$YBO2(M) : \neg top20Movie(M)$$

---

This is now telling that we can get all the information contained in a view called *IMDB*, and the contents of the view is described in terms of the global view predicates. This does not define how this view can be materialized: what must be the input/output to this view. This has to be defined separately using a different language for the capabilities of the resources. Each view is going to be translated to a query (or combination of queries) that can be posed to a specific source.

Given now a conjunctive query as before, the task is now deciding which views to execute. If multiple views contain the same information, that we can choose a subset, assuming that the view definition describes precisely the limitations of the query with respect to the domain.

Recall that the TimesUnion site contains information about the Capitol region only. To represent this, we must be able to represent capitol region in the global data model. Otherwise, the view definition is going to be incomplete. Suppose we have relations *theater*(*T*, *Z*) which tells me that theater *T* is in zipcode *Z*, and *showTime*(*M*, *R*, *T*) where *R* is a show time for movie *M* at theater *T*. Then, we can write a view like:

---

$$TU(M, R) : \neg theater(T, Z), showTime(M, R, T), ZIN(12159, 12160, 12161)$$

---

This view is a bit misleading. First, it does not list all the zipcodes (since I do not know them). Furthermore, it is disjunctive (the IN predicate is an OR) which should be expanded to multiple statements. Of course, we might only care a zip code is in Capital Region without ever needing the get the zip code. Assume another relation in the global model called *region*(*Z*, *X*) which tells with named region *X* a specific zipcode *Z* is in. So, we can now refine this as:

---

$$TU(M, R) : \neg theater(T, Z), showTime(M, R, T), region(Z, capitolregion)$$

---

Note that we never had a way of getting the zipcode from the source. We only needed to express that it lies in the Capitol Region.