CompSci 516
Data Intensive Computing Systems

Lecture 23
Data Integration

Instructor: Sudeepa Roy
Announcements

• No class next week
  – thanksgiving recess!
  – We meet again on 11/30 (Wed)

• Final report first draft due on 11/28 (Mon) night
  – but can update until Friday 12/2 night
  – send me an email if you update

• I will post a message on piazza looking for three groups who will present on 11/30 (Wed)
  – the remaining seven groups present on 12/2 (Fri)
  – 10 minutes talk/demo for each group (8 mins talk + 2 mins questions)
Today’s topic

• An overview of data integration

• Some optional additional slides at the end
Optional Reading:

- The “Principles of Data Integration” book by AnHai Doan, Alon Halevy, Zack Ives
  The lecture slides are based on Ch. 1, 3, 5 of this book

- Data integration PODS 2005 tutorial by Phokion Kolaitis
  (more on the theoretical aspects)
What is Data Integration? 1/2

• Internet and WWW have revolutionized people’s access to digital data
• We take it for granted that a search query into a browser taps into millions on documents and databases and returns what we are looking for
• Systems on the Internet must efficiently and accurately process and serve a large amount of data
What is Data Integration? 2/2

• Unlike traditional RDBMS, the new services need the ability to
  – Share data among multiple organizations
  – Integrate data on a flexible and efficient fashion

• Data integration:
  – A set of techniques that enable building systems geared for flexible sharing and integration of data across multiple autonomous data providers.
Why data integration? 1/2

• With issues like normalizations, and trade-offs in design choices, different people design different schemas for the same data
• Sometimes different needs as well
  – not all attributes are needed by all people
• Sometimes people want to share their data
  – collaborators
  – researchers who want to publish data for others’ use
Why data integration? 2/2

• In the Web,
  – Many websites posting job applications, hotel or flight deals, movie information
  – To keep up with new information and for new need, you may have to look at all of them
  – But now there are websites where you can access all
  – e.g. TripAdvisor helps you see the price of the same hotel on the hotel website, hotels.com, booking.com, expedia, ....

• But this type of data integration has its challenges too
Why data integration? 2/2

• In the Web,
  – Many websites posting job applications, hotel or flight deals, movie information
  – To keep up with new information and for new need, you may have to look at all of them
  – But now there are websites where you can access all
  – e.g. TripAdvisor helps you see the price of the same hotel on the hotel website, hotels.com, booking.com, expedia, ....

• But this type of data integration has its challenges too
Challenges in Data Integration:

1. Query

• Offer uniform access to a set of autonomous and heterogeneous data sources

• Query:
  – query disparate data sources, sometimes update them
Challenges in Data Integration:

2. Number of sources

- Number of sources:
  - challenging even for 10 or 2 data sources
  - amplified for hundreds of sources say in Web-scale
Challenges in Data Integration: 3. Heterogeneity

• Heterogeneity:
  – data sources were developed independently of each other
  – databases, files, html
  – different schema and references
  – some structured some unstructured
Challenges in Data Integration: 4. Autonomy

• Autonomy:
  – the sources may not belong to the same administrative entity
  – even then may be run by different organizations
  – may not have full access to the data
  – there may be privacy concerns
  – the sources may change their formats and access patterns at any time without notifying
Virtual Data Integration Architecture

- Three components
  - Data sources
  - Wrappers
  - Mediated Schema

Diagram:

- Mediated Schema
- Wrappers
- RDBMS\(_1\)
- RDBMS\(_2\)
- HTML
- XML

Duke CS, Fall 2016

CompSci 516: Data Intensive Computing Systems
Virtual Data Integration Architecture

- **Data sources**
  - can be any data model like relational dbms with SQL interface
  - XML with Xquery interface
  - HTML
Virtual Data Integration Architecture

- Wrappers
  - programs that send queries to a data source
  - receives answers
  - apply some basic transformations

- e.g. to a web form source
  - translate query to a http request with a url
  - when the answer comes back as an html file, extract tuples
Virtual Data Integration Architecture

- **Mediated schema**
  - built for the data integration application
  - contains only the aspects that are relevant
  - may not contain all attributes
  - does not store any data typically
  - logical schema for posing queries by the users
Source Descriptions

- Specify the property of the sources that the system needs to know
- main components are semantic mappings
  - relate the schema of the sources to the attributes in the mediated schema
- specified declaratively
- between data sources and mediated schema
  - not between two sources
- also specifies
  - whether sources are complete or not
  - limited access patterns to sources
• source S2 may not contain all the movie showing times in the entire country
• source S3 may be known to contain all movie showing times in New York
• in order to get an answer from the source S1, there needs to be an input for at least one of its attributes

**Example**

<table>
<thead>
<tr>
<th>Movie: Title, director, year, genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors: title, actor</td>
</tr>
<tr>
<td>Plays: movie, location, startTime</td>
</tr>
<tr>
<td>Reviews: title, rating, description</td>
</tr>
</tbody>
</table>

**Movies**
- name, actors, director, genre

**Cinemas**
- place, movie, start

**Cinemas in NYC**
- cinema, title, startTime

**Cinemas in SF**
- location, movie, startingTime

**Reviews**
- title, date, grade, review

---

S1
S2
S3
S4
S5
Example: Query on the Mediated Schema

```
SELECT title, startTime
FROM Movie, Plays
WHERE Movie.title = Plays.movie AND
location="New York" AND
director="Woody Allen"
```

Movies: name, actors, director, genre
Cinemas: place, movie, start
Cinemas in NYC: cinema, title, startTime
Cinemas in SF: location, movie, startingTime
Reviews: title, date, grade, review

Movies

- show times of movies in NYC directed by Woody Allen
Example: Reformulation on source databases: 1/5

- S2: may not contain all the movie showing times in the entire country
- S3: known to contain all movie times in NYC
- S1: to get an answer there needs to be an input for at least one of its attributes

```
SELECT title, startTime
FROM Movie, Plays
WHERE Movie.title = Plays.movie AND location="New York" AND director="Woody Allen"
```

- Tuples for Movie can be obtained from source S1
  - but the attribute title needs to be reformulated to name

**Movie**: Title, director, year, genre  
**Actors**: title, actor  
**Plays**: movie, location, startTime  
**Reviews**: title, rating, description
Example: Reformulation on source databases: 2/5

- Tuples for Plays can be obtained from either source S2 or S3
  - Since the latter is complete for showings in NYC, we choose it over S2

SELECT title, startTime
FROM Movie, Plays
WHERE Movie.title = Plays.movie AND
  location="New York" AND
director="Woody Allen"

Movie: Title, director, year, genre
Actors: title, actor
Plays: movie, location, startTime
Reviews: title, rating, description
Example: Reformulation on source databases: 3/5

- Source S3 requires the title of a movie as input
  - but such a title is not specified in the query
  - the query plan must first access source S1
  - then feed the movie titles returned from S1 as inputs to S3

SELECT title, startTime
FROM Movie, Plays
WHERE Movie.title = Plays.movie AND location="New York" AND director="Woody Allen"

**Movie:** Title, director, year, genre  
**Actors:** title, actor  
**Plays:** movie, location, startTime  
**Reviews:** title, date, grade, review
Example: Reformulation on source databases: 4/5

- Options of logical query plan:
  - access S1, S3
  - could access S1 then S2 as well (possibly not complete)

- Then query optimization
  - as in traditional database system
  - take a logical plan output a physical plan

SELECT title, startTime
FROM Movie, Plays
WHERE Movie.title = Plays.movie AND location="New York" AND director="Woody Allen"

Movie: Title, director, year, genre
Actors: title, actor
Plays: movie, location, startTime
Reviews: title, rating, description

<table>
<thead>
<tr>
<th>Movies</th>
<th>Cinemas</th>
<th>Cinemas in NYC</th>
<th>Cinemas in SF</th>
<th>Reviews</th>
</tr>
</thead>
<tbody>
<tr>
<td>name, actors, director, genre</td>
<td>place, movie, start</td>
<td>cinema, title, startTime</td>
<td>location, movie, startingTime</td>
<td>title, date, grade, review</td>
</tr>
</tbody>
</table>
Example: Reformulation on source databases: 5/5

- Then query execution
  - execute the physical query plan
  - May ask the optimizer to reconsider the plan (unlike RDBMS), e.g. if S3 is too slow

- sometimes contingencies are included in original plan
  - tradeoff between complexity of plan and ability to respond to unexpected events

```
SELECT title, startTime
FROM Movie, Plays
WHERE Movie.title = Plays.movie AND
  location="New York" AND
director="Woody Allen"
```

Movie: Title, director, year, genre
Actors: title, actor
Plays: movie, location, startTime
Reviews: title, rating, description

S1: to get an answer there needs to be an input for at least one of its attributes

S2: may not contain all the movie showing times in the entire country
S3: known to contain all movie times in NYC
S1: sometimes contingencies are included in original plan

S2: may not contain all the movie showing times in the entire country
S3: known to contain all movie times in NYC
S1: to get an answer there needs to be an input for at least one of its attributes

S4: Cinemas in SF
  - location, movie, startingTime

S5: Reviews
  - title, date, grade, review
Schema Mapping should handle the discrepancies between source and the mediated schema: 1/4

• Relation and attribute names
  – “description” in the mediated schema (MS) the same as “review” in S5
  – “name” of Actors in MS and is S3 in NYCCinemas are not the same

---

**Mediated schema**

S1:
- **Actor** (AID, firstName, lastName, nationality, yearOfBirth)
- **Movie** (MID, title), **ActorPlays** (AID, MID)
- **MovieDetail** (MID, director, genre, year)

S2:
- **Cinemas** (place, movie, start)

S3:
- **NYCCinemas** (name, title, startTime)

S4:
- **Reviews** (title, date, grade, review)

S5:
- **MovieGenres** (title, genre)

S6:
- **MovieDirectors** (title, dir)

S7:
- **MovieYears** (title, year)

---

**Source schemas**

S1: Actor (AID, firstName, lastName, nationality, yearOfBirth)
- Movie (MID, title), ActorPlays (AID, MID)
- MovieDetail (MID, director, genre, year)

S2: Cinemas (place, movie, start)

S3: NYCCinemas (name, title, startTime)

S4: Reviews (title, date, grade, review)

S5: MovieGenres (title, genre)

S6: MovieDirectors (title, dir)

S7: MovieYears (title, year)
Schema Mapping should handle the discrepancies between source and the mediated schema: 2/4

• Tabular organization
  – In MS, Actor stores movie title and actor name
  – In S1, a join is needed that has to be specified by the mapping
Schema Mapping should handle the discrepancies between source and the mediated schema: 3/4

- **Domain coverage**
  - the coverage and level of detail may differ
  - S1 stores more info about actors than in MS
Schema Mapping should handle the discrepancies between source and the mediated schema: 3/4

• **Data level variations**
  – GPA as a letter grade vs. a numeric score of 4.0 scale
  – S1 stores actor names in two columns, MS stores in one

<table>
<thead>
<tr>
<th>Movie: Title, director, year, genre</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong>: title, name</td>
</tr>
<tr>
<td><strong>Plays</strong>: movie, location, startTime</td>
</tr>
<tr>
<td><strong>Reviews</strong>: title, rating, description</td>
</tr>
</tbody>
</table>

Mediated schema

<table>
<thead>
<tr>
<th>S1:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong> (AID, firstName, lastName, nationality, yearof Birth)</td>
</tr>
<tr>
<td><strong>Movie</strong> (MID, title), <strong>ActorPlays</strong> (AID, MID)</td>
</tr>
<tr>
<td><strong>MovieDetail</strong> (MID, directorm genre, year)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S2:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cinemas</strong> (place, movie, start)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S3:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NYCCinemas</strong> (name, title, startTime)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S4:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reviews</strong> (title, date, grade, review)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S5:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MovieGenres</strong> (title, genre)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S6:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MovieDirectors</strong> (title, dir)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S7:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MovieYears</strong> (title, year)</td>
</tr>
</tbody>
</table>
Three Desired Properties of Schema Mapping Languages 1/3

• Flexibility
  – significant differences between disparate schemas
  – the languages should be very flexible
  – should be able to express a wide variety of relationships between schemas
Three Desired Properties of Schema Mapping Languages 2/3

• Efficient reformulation
  – our goal is to use the schema mapping to reformulate queries
  – we should be able to develop reformulation algorithms whose properties are well understood and are efficient in practice
  – often competes with flexibility, because more expressive languages are typically harder to reason about
• Easy update
  – for a formalism to be useful in practice, it needs to be easy to add and remove sources
  – If adding a new data source potentially requires inspecting all other sources, the resulting system will be hard to manage for a large number of sources
Three standard schema mapping languages

1. Global-as-View (GAV)
2. Local-as-View (LAV)
3. Global-Local-as-View (GLAV)
Global-as-View (GAV)

• GAV defines the mediated schema (MS) as a set of views over the data sources
  – Mediated Schema = Global schema
  – An intuitive approach

• Mediated schema (MS) G
  – $G_i = \text{some relation in } G$
  – $X_i$ denotes attributes in $G_i$

• Source schema $S_1, S_2, ..., S_n$
GAV Definition

• A GAV schema mapping $M$ is a set of expressions of the form:

• $G_i(X_i) \subseteq Q(S_1, S_2, ..., S_n)$
  – open world assumption
  – instances computed for MS are assumed to be incomplete

• or, $G_i(X_i) = Q(S_1, S_2, ..., S_n)$
  – closed world assumption
  – instances computed for MS are assumed to be complete
GAV Example

- Movie(title, director, year, genre) \supseteq S1.Movie(MID, title), 
  S1.MovieDetail(MID, director, genre, year)
- Movie(title, director, year, genre) \supseteq S5.MovieGenres(title, genre), 
  S6.MovieDirectors(title, director), 
  S7.MovieYears(title, year)
- Plays(movie, location, startTime) \supseteq S2.Cinemas(location, movie, startTime)
- Plays(movie, location, startTime) \supseteq S3.NYCCinemas(location, movie, startTime)

**Mediated schema**

**S1:** 
Actor (AID, firstName, lastName, nationality, yearof Birth)  
Movie (MID, title),  
AcrtorPlays(AID, MID)  
MovieDetail (MID, directorm genre, year)

**S2:** 
Cinemas(place, movie, start)

**S3:** 
NYCCinemas(name, title, startTime)

**S4:** 
Reviews(title, date, grade, review)

**S5:** 
MovieGenres(title, genre)

**S6:** 
MovieDirectors(title, dir)
Suppose we have a data source $S8$ that stored pairs of (actor, director) who worked together on movies.

The only way to model this source in GAV is with the following two descriptions that use NULL:

- $\text{Actors}(\text{NULL, actor}) \supseteq S8(\text{actor, director})$
- $\text{Movie}(\text{NULL, director, NULL, NULL}) \supseteq S8(\text{actor, director})$

These descriptions create tuples in the mediated schema that include NULLs in all columns except one.
Discussions: GAV 2/2

• If the source S8 includes the tuples (Keaton, Allen) and (Pacino, Coppolag), then the tuples computed for the mediated schema would be:
  – Actors(NULL, Keaton), Actors(NULL, Pacino)
  – Movie(NULL, Allen, NULL, NULL), Movie(NULL, Coppola, NULL, NULL)

• Now suppose we have the following query that recreates S8:
  – Q(actor, director) :- Actors(title, actor), Movie(title, director, genre, year)

• We would not be able to retrieve the tuples from S8 because the source descriptions lost the relationship between actor and director.

S8: 
ActTogether(actor, director)

Movie: Title, director, year, genre
Actor: title, name
Plays: movie, location, startTime
Reviews: title, rating, description
Local As View (LAV)

• describes each data source as precisely as possible and independently of any other sources
  – opposite approach to GAV
• Mediated schema (MS) G
• Source schema $S_1, S_2, ..., S_n$
  – $X_i$ denotes attributes in $S_i$
LAV Definition

• A LAV schema mapping $M$ is a set of expressions of the form:

• $S_i(X_i) \subseteq Q_i(G)$
  – open world assumption

• or, $S_i(X_i) = Q_i(G)$
  – closed world assumption
  – but completeness about data sources, not about the MS
LAV Example

- S5.MovieGenres(title, genre) ⊆ Movie(title, director, year, genre)
- S6.MovieDirectors(title, director) ⊆ Movie(title, director, year, genre)
- S7.MovieYears(title, year) ⊆ Movie(title, director, year, genre)
- S8(actor, dir) ⊆ Movie(title, director, year, genre), Actors(title, actor)
- Can also specify constraints on the contents
- S9(title, year, “comedy”) ⊆ Movie(title, director, year, “comedy”), year ≥ 1970

Mediated schema:

- **Movie**: Title, director, year, genre
- **Actor**: title, name
- **Plays**: movie, location, startTime
- **Reviews**: title, rating, description

- S1: Actor (AID, firstName, lastName, nationality, yearOfBirth)
- Movie (MID, title), ActorPlays (AID, MID)
- MovieDetail (MID, director, genre, year)

- S2: Cinemas (place, movie, start)
- S3: NYCCinemas (name, title, startTime)
- S4: Reviews (title, date, grade, review)

- S5: MovieGenres (title, genre)
- S6: MovieDirectors (title, dir)
Global-Local-As-View (GLAV)

• GAV and LAV can be combined into GLAV
• Has the expressive power of both
• The expressions in the schema mapping include
  – a query over the data sources on the left hand side
  – a query on the mediated schema on the right-hand side
• Mediated schema (MS) G
• Source schema $S_1, S_2, ..., S_n$
GLAV Definition

• A GLAV schema mapping \( M \) is a set of expressions of the form:

  • \( Q^S(X) \subseteq Q^G(X) \)
    – open world assumption

  • or, \( Q^S(X) = Q^G(X) \)
    – closed world assumption

• \( Q^G \) is a query over \( G \) whose head variables are \( X \)
• \( Q^S \) is a query over data sources \( S_1, S_2, \ldots, S_n \) where the head variables are also \( S \)
GLAV Example

• Suppose S1 is known to have comedies produced after 1970 only

• S1.Movie(MID, title), S1.MovieDetail(MID, director, genre, year) ⊆ Movie(title, director, "comedy", year), year ≥ 1970
Optional/Additional Slides
Schema Matchings and Mappings

• Specify “matches”, e.g.
  – attribute “name” in one source corresponds to attribute “title” in another
  – “location” is a concatenation of “city, state, zipcode”

• Elaborate matches into semantic “mappings”
  – using queries like SQL
Challenges

• The tasks of creating the matches and mappings are often difficult
  – they require a deep understanding of the semantics of the schemas of the data sources and of the mediated schema
  – This knowledge is typically distributed among multiple people
  – these people are not necessarily database experts and may need help

• There is no algorithm that will take two arbitrary database schemas and flawlessly produce correct matches and mappings
  – goal is to create tools that reduce the time by giving suggestions to the designer
### Two database schemas

<table>
<thead>
<tr>
<th>DVD-VENDOR</th>
<th>AGGREGATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Movies</strong> (id, title, year)</td>
<td><strong>Items</strong> (name, releaseInfo, classification, price)</td>
</tr>
<tr>
<td><strong>Products</strong> (mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)</td>
<td></td>
</tr>
<tr>
<td><strong>Locations</strong> (lid, name, taxRate)</td>
<td></td>
</tr>
</tbody>
</table>

- Attributes and tables in a schema are called its elements.
- The aggregator is not interested in all the details of the product, but only in the attributes that are shown to its customers.
- Schema DVD-VENDOR has 14 elements
  - 11 attributes (e.g., id, title, and year) and three tables (e.g., Movies)
- Schema AGGREGATOR has five elements
  - four attributes and one table.
Semantic mapping

- A query expression that relates a schema $S$ with a schema $T$
  - recall GAV, LAV, GLAV

**DVD-VENDOR**
- Movies(id, title, year)
- Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)
- Locations(lid, name, taxRate)

**AGGREGATOR**
- Items(name, releaseInfo, classification, price)
Semantic mapping - Example 1

- “the title of Movies in the DVD-VENDOR schema is the name attribute in Items in the AGGREGATOR schema.”

- SELECT name as title
- FROM Items
Semantic mapping - Example 2

• “get the price attribute of the Items relation in the AGGREGATOR schema by joining the Products and Locations tables in the DVD-VENDOR schema.”

• SELECT ( basePrice * (1 + taxRate ) ) AS price
• FROM Products , Locations
• WHERE Products . saleLocID = Locations . lid
Semantic mapping - Example 3

- "Get the entire tuple in Items table from DVD-VENDOR"

- \texttt{SELECT title AS name, releaseDate AS releaseInfo, rating AS classification, basePrice * (1 + taxRate) AS price}
- \texttt{FROM Movies, Products, Locations}
- \texttt{WHERE Movies .id = Products .mid AND Products .saleLocID = Locations .lid}
Semantic Matches

• Relates a set of elements in schema S to a set of elements in schema T
  – without specifying the details of the nature of relationship (as SQL queries)
Why are matching and mapping difficult?

• The semantics may not be fully captured in the schemas
  – “rating” may imply movie rating, customer rating, etc
  – sometimes accompanied by English text, hard for systems to parse and understand

• Schema clues can be unreliable
  – two elements may have the same name but different meaning, like “name” or “title”

• Semantics can be subjective
  – what “plot-summary” means
  – sometimes a committee of experts vote

• Combining data may be difficult
  – need to figure out a join path
  – full/left/right outer join or inner join
  – may need filter conditions
  – the designer has figure these out inspecting a large amount of data
  – erroneous and labor prone
Components in a schema matching system

- Matchers
- Combiners
- Constraint Enforcers
- Match Selectors
1. Matchers

- schemas $\rightarrow$ similarity matrix
- takes two schemas S and T as input
- outputs a similarity matrix
- assigns to each element pair $s$ of S and $t$ of T a number between 0 and 1
  - higher the number, $s$ and $t$ are more similar
- e.g.
  - name $\approx$ <name: 1, title: 0:5>
  - releaseInfo $\approx$ <releaseDate: 0:6, releaseCompany: 0.4>
  - price $\approx$ <basePrice: 0:5>
Types of Matchers

• Name-based matchers
  – compares the names of elements
  – but almost never written the same way
  – uses techniques for string matching as edit distance, Jaccard measure etc.; synonyms; normalization (capital letters); hyphens; etc

• Instance-based matchers
  – Look at data instances, builds recognizers (dictionaries), computes overlaps, classification
2. Combiners

- matrix $\times \ldots \times$ matrix $\rightarrow$ matrix

- merges the similarity matrices output by the matchers into a single one

- can take the average, minimum, maximum, or a weighted sum of the similarity scores
Types of Combiners

• Average combiners:
  – Suppose k matchers to predict the scores between the element $s_i$ of schema S and the element $t_j$ of schema T
  – then an average combiner will compute the score between these two elements as the average from these k matchers

• Hand-crafted scripts
  – e.g. if $s_i = \text{address}$, return score of naïve-bayes classifier, else average

• Weighted combiner
  – gives weights to each matcher
3. Constraint Enforcers

- matrix $\times$ constraints $\rightarrow$ matrix

- In addition to clues and heuristics, domain knowledge plays an important role in pruning candidate matches
  - e.g. knowing that many movie titles contain four words or more, but most location names do not, can help us guess that Items.name is more likely to match Movies.title than Locations.name

- an enforcer enforces such constraints on the candidate matches
  - it transforms the similarity matrix produced by the combiner into another one that better reflects the true similarities
Types of Constraint Enforcers : 1/2

• There may be hard or soft domain integrity constraints

• Hard constraints must be applied
  – The enforcer will not output any match combination that violates them

• Soft constraints are of more heuristic nature, and may actually be violated in correct match combinations
  – the enforcer will try to minimize the number (and weight) of the soft constraints being violated
  – but may still output a match combination that violates one or more of them

• Formally, we attach a cost to each constraint
  – For hard constraints, the cost is 1
  – for soft constraints, the cost can be any positive number.
Types of Constraint Enforcers : 2/2

• e.g.
  – c1: If A Items.code, then A is a key (weight = infinity)
  – c2 If A Items.desc, then any random sample of 100 data instances of A must have an average length of at least 20 words (weight = 1.5)
  – c3: If more than half of the attributes of Table U matches those of Table V, then U is similar to V (weight = 1)
4. Match Selectors

• matrix $\rightarrow$ matches

• Produces matches from the similarity matrix output by the constraint enforcer

• name $\approx$ <title : 0:5>
• releaseInfo $\approx$ <releaseDate : 0:6>
• classification $\approx$ <rating : 0:3>
• price $\approx$ <basePrice : 0:5>
• Given the threshold 0.5, the match selector produces matches:
  – name $\approx$ title, releaseInfo $\approx$ releaseDate, and price $\approx$ basePrice
Types of Match Selectors

• The simplest selection strategy is thresholding
  – all pairs of schema elements with similarity score exceeding a given threshold are returned as matches

• More complex strategies include formulating the selection as an optimization problem over a weighted bipartite graph