Semantic Data Integration

Pizza Schemas

Tom Prantz, Celestino Madera Castro, Daniel Lenhardt

Friedrich Schiller University

February 19, 2021

Table of Contents

Description of the Use-Case Scenario

The use-case we are about to describe is the following:

There are two pizza shops, where user and pizza data is stored. There is an app in development, where you can order in either the first store or in the other one. But of course the stores use different systems to manage the data. These Systems are described as Datasources *hereinafter. We don't want the differences in the presentation layer or in higher layers of the app. What we want is one schema we can deal with.*

Description of the Datasources

- Two schemes: S1 and S2
- S1:

StoneOvenPizza(SOPID, PizzaName, nurishment, total)
Customer(CID, PIID, FirstName, LastName)
PaymentInfos(PIID, CreditCardNumber, ccv, expDate)
Order(OID, CID, orderCode, totalAmount, Adress)
OrderItem(PID, OID)

- S2:

Pizza(PID, title, nurishment, price)
User(UID, FirstName, LastName, CCNumber, ccv, expDate)
Order(OID, UID, orderNumber, Sum, AID)
InOrder(OID, PID)
Adress(AID, UID, street, city, zip)

The Mediated Schema

- mediated schema:

User(UID, name, CCInfo)
OrderItem(PID, OID)
Order(OID, UID, orderNumber, totalCost, Adress)
Pizza(PID, pizzaName, nutrition, price)

- unified representation
- **subset** of the data available in the Datasources
- **subset** of the data needed for a real app
- Heterogeneities of the Datasources are mapped in one here (we will describe these later)
- given by the conjuncted queries we want to execute
- these have the formal representation like:

Q(X,T) :-Interview(X,D,Y,H,F), EmployeePerf(E,Y,T,W,Z) where Y is the join variable

Conjuncted Queries

getTotalAmountFromUser(name, totalAmount):- User(UID, name), Order(UID, totalAmount)

- total Costs of the Order from the User
- use our User relation and join this with the Order relation over UID

Conjuncted Queries (2)

userOrder(OID, name):- order(OID, UID), user(name, UID)

- user Name which is related to the user
- use our User relation and join this with the Order relation over UID

Conjuncted Queries (3)

getUserCc(CCInfo, name):- user(name, CCInfo)

- Credit-card information to a user
- information is already in the user relation so we don't need a join over a other relation

Conjuncted Queries (4)

getAdressFromUser(name, Adress):- user(name, UID), order(UID, Adress)

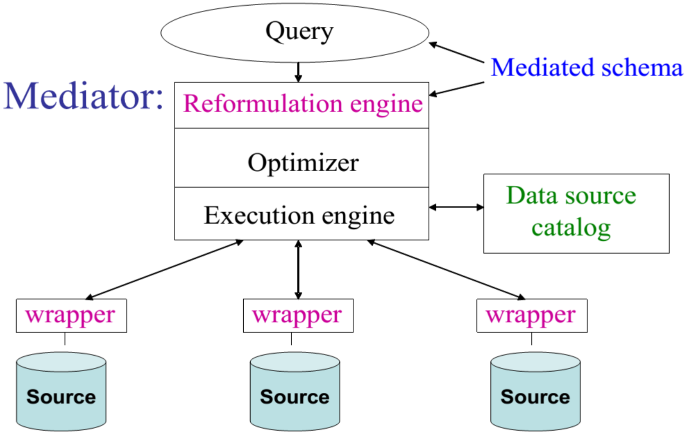
- need to join the user relation with the Order relation because only there the address is saved

Conjuncted Queries (5)

getPizzaFromOrder(orderNumber, pizzaName):- order(OID, orderNumber), orderItem(OID, PID), pizza(PID, pizzaName)

- join order over orderItem and then join this with pizza

Virtual Data Integration



Virtual Data Integration(cont.)

- Virtual integration yields fresh data
- VDI relies very heavy on the external Datasources
- Working on an mediated schema might me more complex, than warehousing
- need to write wrappers and schema matchers
- we will only cover matching

Heterogenities

Two types of schema heterogenities:

- data heterogenities
- schema heterogenities

Three types of schema heterogenities

1. Heterogenities where only the name differs to the sources
2. Heterogenities where multiple attributes in 1 schema relate to one attribute in the other schema. or the other way round
3. and we have heterogenities where one relation is related to multiple relations

Heterogenities in our project

Type 2 & 3:

- S1 to mediated Schema:
  - CreditCardNumber, ccv, expDate in PaymentInfos is CCInfo in mediated (type 3.)
  - first and last name in Customer is name in mediated (type 2.)
- S2 to mediated Schema:
  - first and last name in User is name in mediated (type 2.)
  - street, city, zip in Address is address in mediated (type 3.)

Heterogenities in our project(cont.)

Named heterogenities type 1.

- S1 to mediated Schema:
  - StoneOvenPizza Class is Pizza Class in mediated
  - Customer Class is User Class in medited
  - nurishment in Pizza is nutrition in mediated
  - total in Pizza is price in mediated
  - totalAmount in order is totalCost in mediated
  - orderCode in order is orderNumber in mediated
- S2 to mediated Schema:
  - InOrder Class is OrderItem Class in mediated
  - nurishment in Pizza is nutrition in mediated
  - sum in order is totalCost in mediated
  - titel in Pizza is pizzaName in mediated

Problem definition

- Creating source descriptions requires schema mapping to refer tables and their attributes between mediated schema and datasources
- Using of semantic matches for elaboration into schema mapping
- Three types of matches
  - One-to-one matches e. g. S2.Pizza(title) is matched to Pizza(pizzaName)
  - One-to-many matches e. g. S1.User(FirstName, LastName) is matched to User(name)
  - Many-to-many matches no example in our scenario
- Challenge of reconciling heterogenities with good accuracy and high scalability

Edit-Distance-Matcher

Definition

Sequence-based matcher which compares two strings to find pairs which refer the same real-world-entity by minimal cost for transformation of string x to string y by the function d(x,y) as distance between two strings

- Transformations are deleting, inserting and substituting of single characters
- Equal costs of 1 for every operation of transformation
- Smaller values for higher similarity

Example:
x = “orderNumber” (from mediated schema)
y = “orderCode” (from Source 1)
d(x,y) = 6
Converting distance measure into similarity measure by using the following equation:

$$s(x, y) = 1 - d(x, y)/[max(length(x), length(y))]$$

Example

$$s(x, y) = 1 - 6/11 = 0.54545454545$$

Jaro-Matcher

Definition

Sequence-based matcher which finds common characters and their position in both strings. Common characters without same position marked as transposition.

Computing of similarity measure by using the following equation:

$$jaro(x,y) = 1/3[c/|x| + c/|y| + (c–t/2)/c]$$

With c as total number of common characters and t as total number of transpositions

Example:

- x and y same as in previous slide
- $c = 6$ and $t = 1$

$$jaro(x,y) = 1/3 * (6/11 + 6/9 + (6 - 1/2)/6) = 0.7095959595959596$$

Jaccard-Matcher

Definition

Set-based matcher which tokenize strings in a set of tokens.

B_x as set of generated tokens for x
B_y as set of generated tokens for y
Computing of similarity measure by using the following equation:

$$J(x, y) = |B_x \cap B_y| / |B_x \cup B_y|$$

Example:

$$B_x = \{\#o, or, rd, de, er, rN, Nu, um, mb, be, r\#\}$$

$$B_y = \{\#o, or, rd, de, er, rC, Co, od, e\#\}$$

$$J(x, y) = 5/15 = 0.3333333333333333$$

Combining match predictions

Simplest form of combining via returning average, minimum or maximum of scores from each matcher

Example:

- Given the measures of all three matchers for x and y

$$s(x, y) = 0.54545454545$$

$$jaro(x, y) = 0.7095959595959596$$

$$J(x, y) = 0.3333333333333333$$

- Combining via minimum combination strategy will output 0.3333333333333333
- Combining via maximum combination strategy will output 0.7095959595959596
- Combining via average combination strategy will output 0.5294612794586532

Match selection by thresholding

Thresholding as simplest strategy to choose matches
All pairs with score not less than thresholds are matches

Example:

- Using threshold 0.3 will output (x,y) as match for all combining strategies
- Using threshold 0.5 will output (x,y) as match for average and maximum combining strategies
- Using threshold 0.7 will output (x,y) as match only for maximum combining strategy

Evaluation of semantic matching

- Applying all three combination strategies with thresholds 0.3, 0.5 and 0.7 results in 9 different match-results
- Comparing of this results with manually created solution pattern
- Defining of true positive (TP), false positive (FP), true negative (TN) and false negative (FN) matches for each result
  - Matches of attribute pairs in both pattern as true positive und mismatches in both aus true negative
  - Matches in solution pattern and mismatch in selection result as false negative
  - Mismatches in solution pattern and match in selection result as false positive
- Computing precision by using the following equation:

$$precision = TP/(TP + FP)$$

- Computing recall by using the following equation:

$$recall = TP/(TP + FN)$$

Results

	TP	TN	FN	FP	Recall	Precision
Min - tshld 0.3	9	453	27	85	0.0957	0.25
Min - tshld 0.5	4	470	32	68	0.0555	0.111
Min - tshld 0.7	0	538	36	0	0	0.0
Max - tshld 0.3	36	0	0	538	0.0627	1.0
Max - tshld 0.5	36	0	0	538	0.0627	1.0
Max - tshld 0.7	32	24	4	514	0.0586	0.888
Avg - tshld 0.3	36	4	0	534	0.0631	1.0
Avg - tshld 0.5	21	393	15	145	0.1265	0.583
Avg - tshld 0.7	0	538	36	0	0	0.0

Data Matching

Until now only schema matching, but for tuples we need Data Matching

- check if one tuple in one table matches a tuple in another table
- it is true if they refer to the same real world entity
- the schema is already matched.
- string matching not enough because it could treat tuple as string
- but hard to apply sophisticated techniques and domain-specific knowledge
- example: phone number matches a person exactly (better to keep fields apart)

One approach: rule based matching

Rule-based Matching

Definition

The developer writes rules that specify when two tuples match.

Linearly weighted combination

- Idea: computation of the similarity score for each item, this is weighted and summed
- If sum is bigger than certain threshold the tuples are matched
- CON: some items have enough similarity after threshold to match, but contribute to a higher similarity
- $sim(x, y) = \sum_{i=0}^{n} a_i * sim_i(x_i, y_i)$

Logistic regression

- Solves the problem of item similarity threshold, by logistic sum
- stability is increased
- $sim(x, y) = 1/(1 + e^{-}z)$ where $z = \sum_{i=0}^{n} a_i * sim_i(x_i, y_i)$
- Especially good if we have much signals that have to be processed

Rule-based Matching(cont.)

Pros:

- easy to start, to implement and it runs very fast

Cons:

- can be labor intensive, it takes a lot of time to write good rules
- it's difficult to find the right weights

The weights can be learned, which is especially good for very big schema (learn based matching).
A example for our project would be:
Table *S1.StoneOvenPizza(SOPID, PizzaName, nurishment, total)* with tuples x and table **S2.Pizza(PID, title, nurishment, price)** with tuples y
mediated as Pizza(PID, pizzaName, nutrition, price)

$$sim(x, y) = 0.01 * s_{PID}(x, y) + 0.5 * s_{pizzaName}(x, y) + 0.3 * s_{nutrition}(x, y) + 0.19 * s_{price}(x, y)$$

Thanks for listening