Information Integration – Exercise 3 – Gabriel Glaser

Task 1: Mediator-Wrapper Architecture

a) **What data needs to be extracted** from the sources. Ensure that these data are necessary, i.e., do not extract any data that is not relevant for producing the desired query result. b) Implement the **wrapper for each source**. Your answer should include (i) an **SQL query for D1** and (ii) a loop in **pseudo-code for D2**. c) Implement the **mediator** functionality in **SQL**. d) Assume that source **D1 changes its schema** to (SSN, Name, Birthday, Address). Do any components of the mediator-wrapper architectures need be **changed**? If so, please **adapt your code** from the answers to the questions above. e) Let us now assume we notice that the **third tuple of table 3 is no result needed** by our application, as we are only interested in average salaries where the birth year is known. Do any components of the mediator-wrapper architectures need be **changed**? If so, please **adapt your code** from the answers to the questions above. f) Assume the **integrated schema evolves** to (BirthYear, AverageSalary, State). Do any components of the mediator-wrapper architectures need be **changed**? If so, please **adapt your code** from the answers to the questions above.

a)

- **D1**: Columns *SSN* and *Birthday*
- **D2**: Last two columns

b)

- **Wrapper for D1**:

```
CREATE VIEW WrapperTableD1 AS
  SELECT SSN, Birthday FROM TableD1;
UPDATE WrapperTableD1 SET Birthday = YEAR(Birthday);
ALTER TABLE WrapperTableD1
  RENAME COLUMN Birthday TO BirthYear;
```

- **Wrapper for D2**:

```
# python-like script
run_sql_query(
    CREATE TABLE WrapperTableD2 (
        AverageSalary INT(255),
        SSN CHAR(12)
    );
)
for row in csv_rows:
    cells = row.split(",")
    salary = int(remove_last_char(cells[4]))
    ssn = cells[5]
    run_sql_query(
        INSERT INTO WrapperTableD2
            VALUES(salary, ssn)
        );
    )
```

c) **Mediator**:

```
CREATE VIEW tmp AS
    SELECT * FROM WrapperTableD1, WrapperTableD2
    WHERE WrapperTableD1.SSN = WrapperTableD2.SSN;
CREATE VIEW IntegratedTable AS
    SELECT BirthYear, AverageSalary FROM tmp;
```

d) Nothing has to be changed, because the label of all relevant attributes didn't change. Therefore, the Wrapper for *D1* remains to correctly return a table with birthyears and ssns.

e) This requires a change which, for instance, can be implemented in the mediator by adding another condition to the join of both wrapper tables. The change has the semantic that requires the birthyear being non null, i.e.,

AND WrapperTableD1.BirthYear **IS NOT NULL**.

f) So far, the integrated table doesn't contain the column *State*, thus it has to be added (in the mediator):

ALTER TABLE IntegratedTable **ADD** State **INT**(255);.

Task 2: Stable Marriage Algorithm

a) Construct an example (list of men, women, preferences) in which there is more than one stable matching (you only need two men and two women to do this).

b) Suppose that the men all have different favorite women. How many steps (proposals) does it take for the algorithm to converge?

c) Suppose that the men have identical preferences. How many steps (proposals) does it take for the algorithm to converge?

d) Suppose preferences are given by Table 4 and Table 5. Find a stable matching with men making proposals.

e) Reusing the same preferences as in 2d given in Table 4 and Table 5, find a stable matching with women making proposals.

a)

Men	Women
A: 1, 2	1: A, B
B: 1, 2	2: A. B

- (A,1) & (B,2) is stable, because A doesn't prefer 2 and 1 doesn't prefer B.
- Similarly (B,1) & (A,2) is stable, because B doesn't prefer 2 and 2 doesn't prefer B.

b) The algorithm needs exactly as many steps as men are there. This is because, during the algorithm each man will propose to a woman who didn't have an offer before. Thus, she accepts and also, won't get further proposals which may could lead to a change (break up and new search for the man who lost the woman who he initially proposed to).

c) Let's assume each man m_i has the priority list $(w_1, w_2, ..., w_n)$. It is the case that each man m_i performs at least 1, but at most n proposals. Also, it is not possible that any two men $m_i, m_j, i \neq j$ have the same amount of proposals, because then both of their last proposal would address the same woman (which is not allowed). The only way of distributing the (pair-wise different) number of proposals of n men would be that

- one man has 1 proposal
- one man has 2 proposals
- one man has 3 proposals

- ...
- one man has n proposals

Therefore, there are

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

proposals.

d)

Adam $\rightarrow$ Beth $\surd$	
Bill $\rightarrow$ Diane $\surd$	
Carl $\rightarrow$ Beth $\surd$	Adam loses partner
Adam $\rightarrow$ Amy $\surd$	
Dan $\rightarrow$ Amy $\lightning$	
Dan $\rightarrow$ Diane $\surd$	Bill loses partner
Bill $\rightarrow$ Beth $\lightning$	
Bill $\rightarrow$ Amy $\lightning$	
Bill $\rightarrow$ Cara $\surd$	
Eric $\rightarrow$ Beth $\lightning$	
Eric $\rightarrow$ Diane $\surd$	Dan loses partner
Dan $\rightarrow$ Cara $\lightning$	
Dan $\rightarrow$ Beth $\lightning$	
Dan $\rightarrow$ Ellen $\surd$	

(Each arrow represents a proposal which may be accepted $\surd$ or rejected $\lightning$)
Result: (Adam, Amy), (Bill, Cara), (Carl, Beth), (Dan, Ellen), (Eric, Diane).

e)

Amy $\rightarrow$ Eric $\surd$
Beth $\rightarrow$ Carl $\surd$
Cara $\rightarrow$ Bill $\surd$
Diane $\rightarrow$ Adam $\surd$
Ellen $\rightarrow$ Dan $\surd$

Result: (Amy, Eric), (Beth, Carl), (Cara, Bill), (Diane, Adam), (Ellen, Dan).

Task 3: Schema Matching

a) Determine an optimal schema matching by hand, i.e., **manually** identify the correspondences that compose the **ideal logical mapping**.

b) Given the logical mapping of 3a), describe the different **mapping situations** you observe, similarly to the mapping situation exercise done in class (see Chapter 5, slide 19).

c) We now perform an **"automatic"** schema matching. In the first step you have to calculate the similarity for each pair of (source, target) with the following formula:

```
sim(source, target) = 1 - levenshteinDistance(source, target) /
    maxLength(source, target)
```

The function maxLength(s1,s2) simply returns the maximum length of the two attribute names s1 and s2, e.g., maxLength(name, firstname) = 9. Remove the prefix "person_" from the source schema (thus person_name → name). Furthermore, the Levenshtein function you use should be case-insensitive Explain how the Levenshtein distance is calculated and demonstrate it on one (source, target) example by filling the Levenshtein matrix. Compute the similarity for all other (source, target) pairs as well.

d) Using the pairwise similarities from 3c), determine the **similarity threshold** that **maximizes the f-measure**. To this end, also compute the overall **precision** and **recall** of the mapping for the different thresholds. Draw/Plot your recall / precision / f-measure results in a graph and identify the similarity threshold where the f-measure is maximal.

e) What is the **optimal matching** according to the defined metrics and previous answers?

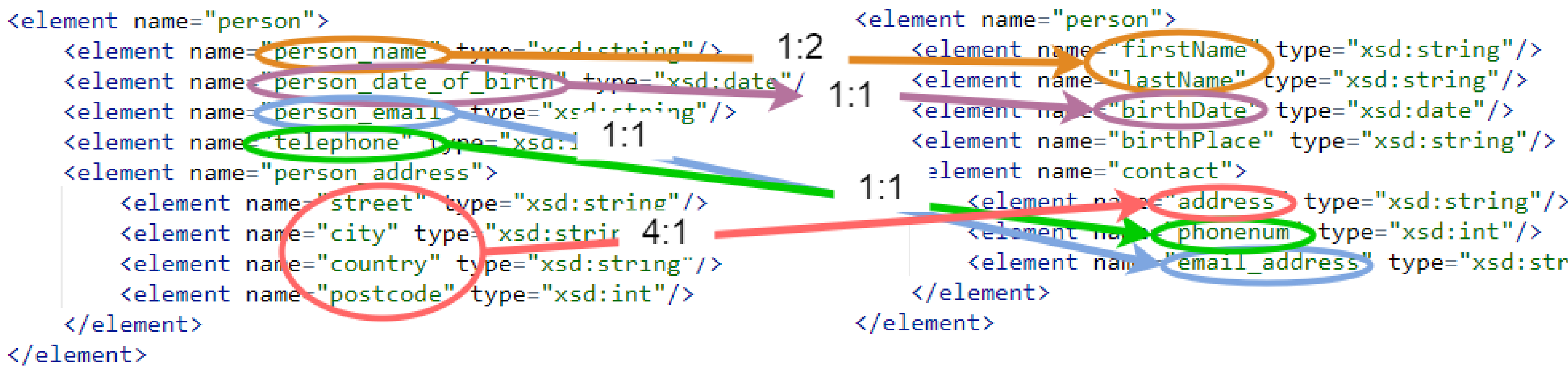
f) According to your expertise, are there **disadvatanges** of these metrics in the given scenario? If so, advertise strategies that are better suited for identifying optimal mappings.

a) Optimal Schema:

```
<element name="person">
    <element name="name" type="xsd:string"/>
    <element name="birthDate" type="xsd:date"/>
    <element name="birthPlace" type="xsd:string"/>
    <element name="address" type="xsd:string"/>
    <element name="phoneNumber" type="xsd:string"/>
    <element name="emailAddress" type="xsd:string"/>
</element>
```

Preserves all information from both schemas while keeping the effort for data fusion low. For instance, *name* from schema 1 can be directly written to an instance of the global schema and *firstName*, *lastName* can be easily transformed accordingly. Similarly, *address* of schema 2 can be directly written and the distinguished address components in schema 1 can be easily transformed accordingly.

b)



c) Illustration of Levenshtein(*phonenum*, *telephone*):

		t	e	l	e	p	h	o	n	e
	0	1	2	3	4	5	6	7	8	9
p	1	1	2	3	4	4	5	6	7	8
h	2	2	2	3	4	5	4	5	6	7
o	3	3	3	3	4	5	5	4	5	6
n	4	4	4	4	4	5	6	5	4	5
e	5	5	4	5	4	5	6	6	5	4
n	6	6	5	5	5	5	6	6	6	5
u	7	7	6	6	6	6	6	7	7	6
m	8	8	7	7	7	7	7	7	8	7

- Each entry of the matrix above represents the number of character-operations (insert, replace, remove) needed to transform the top prefix to the left prefix.
- Therefore, the first row/column can be initialized with 0, 1, 2 etc., because they represent the number of operations to transform the empty string to the respective prefix.
- Furthermore, to transform the prefixes of cell (i, j) into each other, either
  - Take previous shorter prefix (cell above) plus an insert operation.
  - Take longer prefix (cell left) plus a remove operation.
  - Take prefix of same length (cell diagonal top-left) plus a replace operation. The plus one is not necessary, if the characters the the cell already are equal.

- Applying this yields a table like above while the bottom-right value represents the final distance measure.

Therefore, the Levenshtein distance in example is 7.
Thus, $\text{sim}(\textit{phonenum}, \textit{telephone}) = 1 - \frac{7}{\max\{|\textit{phonenum}|,|\textit{telephone}|\}} = 1 - \frac{7}{9} = \frac{2}{9}$
Remaining similarities:

source	target	sim
name	firstName	0.444
name	lastName	0.5
name	birthDate	0.222
name	birthPlace	0.199
name	address	0.142
name	phonenum	0.125
name	email_address	0.153
date_of_birth	firstName	0.0
date_of_birth	lastName	0.076
date_of_birth	birthDate	0.076
date_of_birth	birthPlace	0.0
date_of_birth	address	0.076
date_of_birth	phonenum	0.076
date_of_birth	email_address	0.076
email	firstName	0.111
email	lastName	0.125
email	birthDate	0.111
email	birthPlace	0.099
email	address	0.0
email	phonenum	0.0
email	email_address	0.384

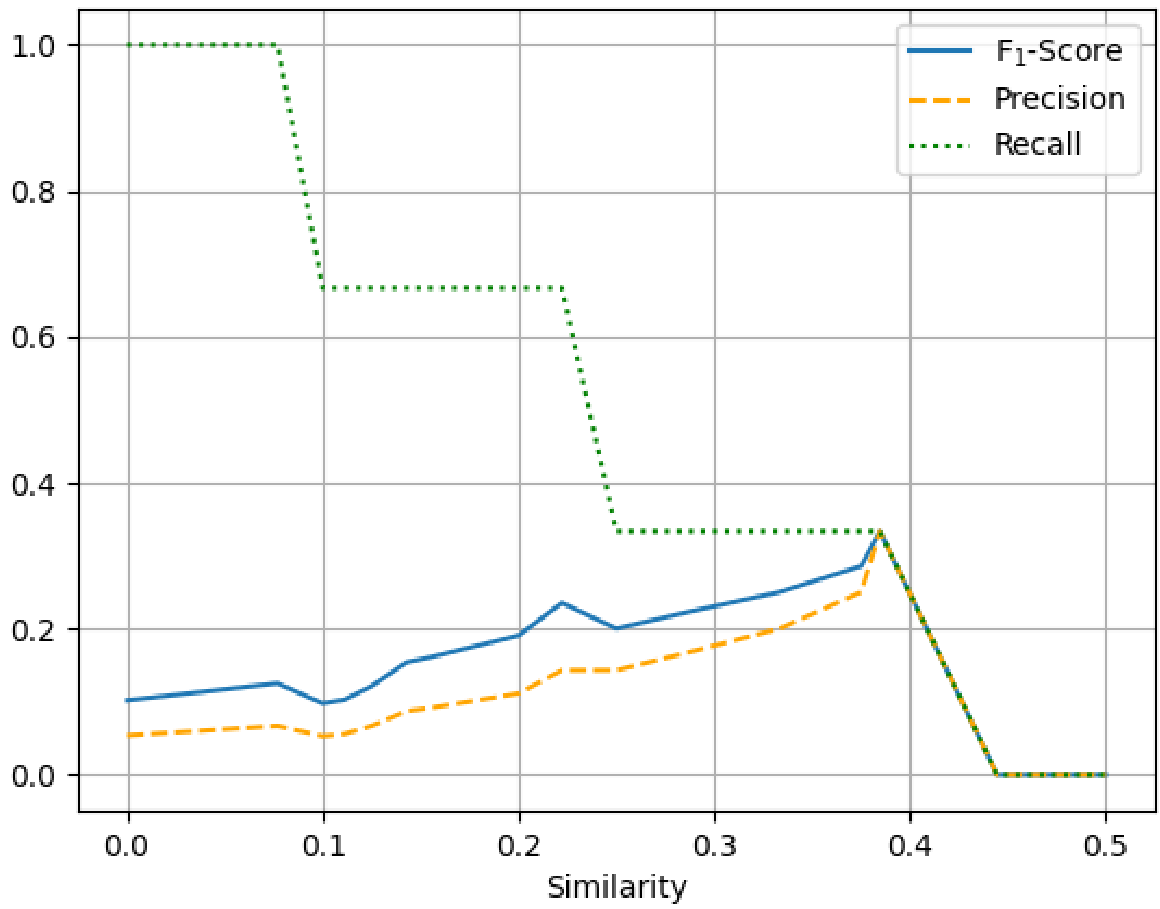
source	target	sim
telephone	firstName	0.111
telephone	lastName	0.111
telephone	birthDate	0.111
telephone	birthPlace	0.199
telephone	address	0.0
telephone	phonenum	0.222
telephone	email_address	0.153
street	firstName	0.222
street	lastName	0.25
street	birthDate	0.222
street	birthPlace	0.099
street	address	0.285
street	phonenum	0.125
street	email_address	0.153
city	firstName	0.222
city	lastName	0.125
city	birthDate	0.222
city	birthPlace	0.199
city	address	0.0
city	phonenum	0.0
city	email_address	0.076

source	target	sim
country	firstName	0.111
country	lastName	0.125
country	birthDate	0.0
country	birthPlace	0.0
country	address	0.0
country	phonenum	0.125
country	email_address	0.076
postcode	firstName	0.333
postcode	lastName	0.375
postcode	birthDate	0.222
postcode	birthPlace	0.199
postcode	address	0.0
postcode	phonenum	0.125
postcode	email_address	0.153

d) Assume ground truth:

- date_of_birth = birthDate
- email = email_address
- telephone = phonenum

Other mappings are considered wrong, because either the mapping is not 1:1 or they have different semantics. Furthermore, all possible thresholds for all similarity values which need to be tested are all different ones retrieved in the task before (which are 16 similarities).



⇒ Best similarity threshold is ≈ 0.39.

	should be mapped	shouldn't be mapped
were mapped	TP	FP
weren't mapped	FN	TN

- *Precision*: $\frac{TP}{TP+FP}$, i.e., proportion of how many mappings were correct.
- *Recall*: $\frac{TP}{TP+FN}$, i.e., proportion of how many mappings were found, in comparison to the number of possible correct mappings.
- $F_1 = \frac{2PR}{P+R}$

e) name = lastName
email = email_address
postcode = firstName (actually, lastName had a higher similarity, but not available for mapping)

f) Using Levenshtein can be bad when equivalent attributes contain substrings at different word position, e.g., "telephone" and "phonenum". An improved strategy could be to check whether one attribute label has a substring longer than, e.g., half or $\frac{1}{3}$ of the word and is part of the other attribute label string.