Name

CWID

Homework Assignment 2

February 16, 2016

CS520

Please leave this empty!

1.1 1.2 1.3 1.4 1.5 1.6

1.7 1.8 1.9 1.10 1.11 1.12 1.13 Sum

Instructions

- Try to answer all the questions using what you have learned in class
- The assignment is not graded
- There is a theoretical and practical part
- **When writing a query, write the query in a way that it would work over all possible database instances and not just for the given example instance!**

Lab Part

- This part of the assignment helps you to practice the techniques we have introduced in class

Hospital Dataset

- We have uploaded a hospital dataset to the course webpage: `http://cs.iit.edu/~cs520/hospital.csv`
- The database instance is stored in a CSV file
- The schema of this database contains a single table with attributes
  - **providernumber**
  - **hospitalname**
  - **address1**
  - **address2**
  - **address3**
  - **city**
  - **state**
  - **zip**
  - **country**
  - **phone**
  - **hospitaltype**
  - **hospitalowner**
  - **emergencyservice**
  - **condition**
  - **measurecode**
  - **measurename**
  - **score**
  - **sample**
  - **stateavg**

The following constraints (functional dependencies) have been defined for the dataset:

$$
\begin{aligned}
&e_0 : zip \rightarrow city \\
&e_1 : zip \rightarrow state \\
&e_2 : phone \rightarrow zip \\
&e_3 : phone \rightarrow city \\
&e_4 : phone \rightarrow state \\
&e_7 : providernumber, measurecode \rightarrow stateavg \\
&e_8 : state, measurecode \rightarrow stateavg
\end{aligned}
$$

Part 1.1 Reuse Hospital Schema and Dataset (Total: 0 Points)

- Recall that in the last assignment we asked you to load the hospital dataset into the DBMS of your choice. In this assignment you will execute several data cleaning task over this dataset.

Part 1.2 Detecting Constraint Violations (Total: 0 Points)

- The dataset is dirty with respect to the functional dependencies defined on the previous page.
- Write SQL queries using the method we have discussed in class to detect which constraints are violated (boolean queries). Execute these queries. Which constraints are violated?
- Modify the queries from the previous step to detect pairs of tuples that a constraint.
- Execute these queries and store their results

Part 1.3 Fixing Violations (Total: 0 Points)

- Pick one constraint with violations and write a program for fixing these violations
  - Your program should use "updates equating the right-hand side" to fix violations
  - Start by computing equivalence classes (if you modify the SQL queries used for detection in a particular fashion this will help you to build these equivalence classes)
  - To fix a violation implement one of the following strategies
    - *Frequency per class*: pick the value that occurs the most within each right-hand side attribute of an equivalence class.
    - *Total frequency*: pick the value that occurs most within each right-hand side attribute (consider all tuples in the database).
    - *Cost of the update*: use the edit-distance measure (see below) you have implemented to compute the cost of updating values. The cost of updating attribute A of tuples from a equivalence class E to a value c defined as $\sum_{t \in E} d_{edit}(t, t')$ where t' is the version of t after updating A to c.

Part 1.4 Edit Distance (Total: 0 Points)

- Implement the edit distance measure in your favorite programming language
- Test it with a few strings (e.g., the examples from the slides)

Part 1.5 Entity Resolution (Total: 0 Points)

- Create a table $medcond(city, state, zip, condition)$ by running a query over the hospital table and store it as a separate table. We will use this table for entity resolution.
- Naturally, this table has a lot of duplicates (same values). Now we randomly update tuples in the database to create near-duplicates by adding or deleting a few characters. E.g., in SQL you can use queries like the one shown below to add one or more random characters to some values of an attribute.

```
UPDATE hospital
SET city = city
    || CASE WHEN random() > 0.9 THEN 'A' ELSE '' END
    || CASE WHEN random() > 0.5 THEN 'B' ELSE '' END
    || CASE WHEN random() > 0.2 THEN 'C' ELSE '' END
WHERE random() > 0.5;
```

- Assign some reasonable weights to the attributes. For example, state should be less predictive than zip code.
- Use edit distance as a similarity metric for all attributes.
- Fix a threshold β.
- Find duplicates using the weighted combination of the edit distance for the attributes of the table.

Part 1.6 Data Fusion (Total: 0 Points)

- For duplicates that you have identified in the previous step, fuse conflicting values for the *state* attribute by choosing the value that is more common in the database, i.e., you would have to run a query upfront to determine the value distribution of the state column.

Part 1.7 The Llunatic cleaning system (Total: 0 Points)

Llunatic is a system for constraint-based data cleaning that is available as open source.

- Download llunatic from the link given on the project page `http://www.db.unibas.it/projects/llunatic/`.
- Download the examples from that webpage and get used to system
- If you like to, then try to use llunatic to repair the hospital dataset

Part 1.8 Use BART to mess up some data (Total: 0 Points)

BART is an open source system for injecting errors into a clean database in a controlled manner. The system was developed to support evaluations of data cleaning algorithms and systems.

- Download BART from the link given on the project page `http://www.db.unibas.it/projects/bart/`.
- Download the examples from that webpage and use the system to create a few dirty datasets
- Create versions with different amount of violations (errors) and test how well they are cleaned by llunatic

Part 1.9 Open end (Total: 0 Points)

There are several other freely available cleaning/preparation solutions available online. Browse the web and try out a few. For example, `http://datacleaner.org` and `http://openrefine.org/`.

Theory Part

- This part of the assignment helps you to practice the techniques we have introduced in class.

Consider the following transportation database schema and example instance:

road

fromCity	toCity	length
Chicago	Evanston	13
Chicago	Evanston	14
Chicago	Oak Park	8
Oak Park	Naperville	20
Chicago	Naperville	18

city

name	gasPrice	population
Chicago	1.80	5,000,000
Evanston	1.9	300,000
Oak Park	1.5	500,000
Naperville	1.6	22,000

train

fromCity	toCity	price
Chicago	Evanston	20
Chicago	Oak Park	34
Oak Park	Naperville	12

Hints:

- Attributes with black background form the primary key of a relation
- The attributes *fromCity* and *toCity* of relation *road* are both foreign keys to relation *city*
- The attributes *fromCity* and *toCity* or relation *trans* are both foreign keys to relation *city*

Part 1.10 Detection Queries (Total: 0 Points)

Question 1.10.1 Translate detection queries (0 Points)

Translate the SQL queries you have written for detecting violations of constraints (e_0 to $_8$) in the lab part of the assignment and translate them into datalog.

Question 1.10.2 Write detection queries for transportation schema constraints (0 Points)

Consider the logical versions of the constraints defined for the transportations schema (see last assignment). Write boolean detection queries for these constraints in Datalog. Note that the detection queries for foreign key constraints require the use of negation in the Datalog queries.

$$
\begin{aligned}
PK(city): &\quad \forall name, gP1, gP2, ppl1, ppl2 : city(name, gP1, ppl1) \wedge city(name, gP2, ppl2) \rightarrow gP1 = gP2 \wedge ppl1 = ppl2 \\
PK(train): &\quad \forall fCity, tCity, p1, p2 : train(fCity, tCity, p1) \wedge train(fCity, tCity, p2) \rightarrow p1 = p2 \\
FK_1(road): &\quad \forall fCity, t, l : road(fCity, t, l) \rightarrow \exists gPrice, ppl : city(fCity, gPrice, ppl) \\
FK_2(road): &\quad \forall f, tCity, l : road(f, tCity, l) \rightarrow \exists gPrice, ppl : city(tCity, gPrice, ppl) \\
FK_1(train): &\quad \forall fCity, t, l : train(fCity, t, l) \rightarrow \exists gPrice, ppl : city(fCity, gPrice, ppl) \\
FK_2(train): &\quad \forall f, tCity, l : train(f, tCity, l) \rightarrow \exists gPrice, ppl : city(tCity, gPrice, ppl)
\end{aligned}
$$

Question 1.10.3 Write detection queries for denial constraints (0 Points)

Consider the denial constraints developed over the transportations schema (see last assignment). Write boolean detection queries for these constraints in Datalog.

$$c_1 : \forall \neg(city(X, Y, Z) \wedge Z > 200,000 \wedge Y < 1.5)$$

$$c_2 : \forall \neg(road(X, Y, L1) \wedge road(X, Y, L2) \wedge abs(L1 - L2) > 10)$$

$$c_3 : \forall \neg(train(X, Y, P) \wedge train(X, Z, P1) \wedge train(Z, Y, P2) \wedge P < P1)$$

$$c_4 : \forall \neg(train(X, Y, P) \wedge train(X, Z, P1) \wedge train(Z, Y, P2) \wedge P < P2)$$

Part 1.11 Query Equivalence and Containment (Total: Points)

Question 1.11.1 (6 Points)

Determine which of the following queries are contained in each other or equivalent to each other under set semantics. Recall that to determine containment you need to check whether there are containment mappings between queries. Write down one containment mapping for each pair of queries in each direction (if it exists) and fill out the table below.

$$Q_1(X,Y) :- R(X,Z), R(A,Y).$$
$$Q_2(X,Y) :- R(X,Y).$$
$$Q_3(X,Y) :- R(X,Z), R(A,Z), R(Y,Z), R(B,Z).$$
$$Q_4(U,U) :- R(U,U).$$
$$Q_5(X,Y) :- R(X,X), R(Y,Y).$$
$$Q_6(X,Y) :- R(X,X), R(X,Z), R(Z,Y), R(Y,Y).$$

$Q \sqsubseteq Q'$	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
Q_1	■					
Q_2		■				
Q_3			■			
Q_4				■		
Q_5					■	
Q_6						■