Name	CWID

Homework Assignment 1

February 2015

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 Create Schema and Load Dataset (Total: 0 Points)

- Load the database into your favorite database / NoSQL store / distributed file system. Use a system you are comfortable with and where you would know how to write the queries required for the next questions (have a look at these questions first).
- As an example here are the steps outlined for Postgres
  - Run the DDL to create a (single table) schema for the dataset
  - Use the loader utility of your database (e.g., `COPY` command in Postgres) to load the content of the CSV file into your table

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 queries using the method we have discussed in class to detect pairs of tuples that violate 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 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.

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 1.7 Datalog (Total: 0 Points)

Question 1.7.1 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 1.7.2 Reachability of cities (0 Points)

Write a datalog program that computes which cities are reachable from each other. To reach a city from another city one has to either take a train connecting these cities or a road. Note that it may require multiple steps to reach one city from another. Furthermore, roads and trains are running in both directions even if the database only contains only one direction. For example, in the example instance there is a train from *Oak Park* to *Chicago*.

Question 1.7.3 Train lines (0 Points)

Write a datalog program that computes which cities are reachable from each other via train with at most 2 transfers.

Question 1.7.4 Train lines (0 Points)

Translate the program from the previous question into relational algebra and SQL

Part 1.8 Constraints (Total: 0 Points)

Question 1.8.1 Translation into logical formalism (0 Points)

Translate the functional dependencies e_0 to e_8 from the lab part into the first-order logical representation that was introduced in class.

Question 1.8.2 Translation into logical formalism (0 Points)

Translate the primary and foreign key constraints of the transportation schema present before into the first-order logical representation that was introduced in class.

Question 1.8.3 Translation into denial constraints (0 Points)

Translate the primary and foreign key constraints of the transportation schema present before into the first-order logical representation that was introduced in class.

Question 1.8.4 Creating denial constraints (0 Points)

Create denial constraints over the transportation schema that encode the following restrictions (note: it may be necessary to use more than one constraint to express some of the restrictions):

1. The gas price of cities with over 200,000 inhabitats (population attribute) is always above or equals to 1.5
2. The difference in length between two roads connecting the same cities is never more than 10 miles
3. The direct train route between two cities is always more expensive than each individual train on a route with one intermediate stop. E.g., the train $(Chicago, Naperville)$ has to be more expensive than the trains $(Chicago, OakPark)$ and $(OakPark, Naperville)$

Part 1.9 Query Equivalence and Containment (Total: Points)

Question 1.9.1 (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.

$$\begin{aligned}
Q_1(X,Y) &: -R(X,Z), R(A,Y).\\
Q_2(X,Y) &: -R(X,Y).\\
Q_3(X,Y) &: -R(X,Z), R(A,Z), R(Y,Z), R(B,Z).\\
Q_4(U,U) &: -R(U,U).\\
Q_5(X,Y) &: -R(X,X), R(Y,Y).\\
Q_6(X,Y) &: -R(X,X), R(X,Z), R(Z,Y), R(Y,Y).
\end{aligned}$$

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