PS2

October 13, 2016

```
In [ ]: %load_ext sql
        %sql sqlite:///
```

1 Problem Set 2

[100 points total]

1.0.1 Instructions / Notes:

Read these carefully

- This problem set does **not** come with a dataset to load; instead, make your own instances of tables, either as solutions to the problems or for testing solutions to the problems.
- You are encouraged to create new IPython notebook cells to use for testing, debugging, exploring, etc. **Just make sure that your final answer for each question is in its own cell when you submit.**
- When you see `In [*]:` to the left of the cell you are executing, this means that the code / query is *running*.
  - **If the cell is hanging (i.e. running for too long): You can restart the SQL connection by restarting the entire python kernel.**
  - To restart the python kernel: Use the menu bar (Kernel >> Restart).
  - To restart the SQL connection: Re-execute the SQL connection cell at the top of the notebook
  - You will also need to restart the connection if you want to load a different version of the database file
- Remember:
  - `%sql [SQL]` is for *single line* SQL queries
  - `%%sql [SQL]` is for *multi line* SQL queries
- **See Piazza for submission instructions**
- *Have fun!*

1.1 Problem 1

[20 points total]

For each part of this problem you will provide a *single* SQL query to check whether a certain condition holds on a specific instance of a relation in the following way: **your query should return**

an empty result if and only if the condition holds on the instance. (If the condition *doesn't hold*, your query should return something non-empty, but it doesn't matter what this is).

Note our language here: the conditions that are specified in each part may not hold **in general** for *any instance* of a relation without knowing the externally-defined functional dependencies of that relation. You are checking whether they *could hold* in general for the relation, given any specific set of tuples.

You may assume that there will be no `NULL` values in the tables, **and you may assume that the relations are *sets* rather than multisets**, but otherwise your query should work for general instances. We define the schemas of the tables used below for convenience, but in this problem **you will need to construct your own test tables if you wish to check your answers!**

```
In [ ]: %%sql
        DROP TABLE IF EXISTS R; CREATE TABLE R (A INT, B INT, C INT, D INT, E INT);
        DROP TABLE IF EXISTS S; CREATE TABLE S (A INT, B INT, C INT);
```

1.1.1 Part (a)

[5 points]

$\{A, B\} \to \{C\}$ and $\{C\} \to \{A, B\}$ hold for a relation $R(A, B, C, D, E)$.

```
In [ ]: %%sql
```

1.1.2 Part (b)

[5 points]

$\{A, B, C\}$ is a **superkey** for a relation $R(A, B, C, D, E)$.

```
In [ ]: %%sql
```

1.1.3 Part (c)

[5 points]

$\{A\}$ and $\{B\}$ are each **keys** for a relation $S(A, B, C)$.

```
In [ ]: %%sql
```

1.1.4 Part (d)

[5 points]

A **multi-valued dependency (MVD)** is defined as follows: let R be a schema i.e. a set of attributes, and consider two *sets of attributes* $X \subseteq R$ and $Y \subseteq R$. We say that a multi-valued dependency (MVD), written:

$X \twoheadrightarrow Y$

holds on R if whenever there are two tuples t_1, t_2 such that $t_1[X] = t_2[X]$, there also exists a third tuple t_3 such that:

- $t_3[X] = t_1[X] = t_2[X]$
- $t_3[Y] = t_1[Y]$
- $t_3[R \setminus Y] = t_2[R \setminus Y]$

Note that $R \setminus B$ is all the attributes in R that are not in B, and that t_3 **need not be distinct from** t_1 **or** t_2. Note especially that an MVD holds on an entire *relation*, meaning that any two tuples (in any order) in the relation should satisfy the above conditions if the MVD holds. **See the end of the lecture 6 slides for more on MVDs!**

In this problem, write your query following the specifications listed at the top of problem 1 to check if the MVD $\{A\} \twoheadrightarrow \{C, E\}$ holds for a relation $R(A, B, C, D, E)$.

```
In [ ]: %%sql
```

1.2 Problem 2

[30 points total]

Consider a relation $R(X, Y, Z)$ with some list of functional dependencies $f_1, f_2, \ldots, f_n$. Suppose that K is a **superkey** for this relation, given these functional dependencies (recall that any superkey K must be a subset of the attributes of R, which are $\{X, Y, Z\}$). In each part of this problem, we will examine what can happen if we add an additional functional dependency to our relation.

To answer **yes**, provide python code that assigns the variable `answer` to `True` and assigns `explanation` to be a python string which contains a (short!) explanation of why. For example, if we are given the statement "if K is a key, then it will still be a superkey when we add a new functional dependency", we can give the following answer:

```
answer = True
explanation = "<insert explanation here>"
```

To answer **no**, provide python code that assigns the variable `answer` to `False`. You must also assign the variable `K` to be a set of attributes, the variable `FDs` (functional dependencies) to be a python list having elements that are *pairs* of sets, and the variable `new_FD` (the new functional dependency to be added) to be a pair of sets, such that these three variables together produce a counter-example for the desired statement. For example, a counter-example to the statement "if $K \to \{Z\}$ is false, then it will remain false when we add a new functional depencency" could look like:

```
answer = False
K = set("X")
FDs = [
    (set("Y"), set("Z")),
    (set(("Y", "Z")), set("X"))
]
new_FD = (set("X"), set("Z"))
```

1.2.1 Part (a)

[10 points]

CS145 student Alex claims that if we add any new functional dependency $f_{n+1} = U \to V$ (where U and V are subsets of $\{X, Y, Z\}$) to our list of functional dependencies, then K will still be a superkey for R given $f_1, f_2, \ldots, f_{n+1}$. Is Alex correct? If yes, explain why. If no, provide a counter-example.

```
In [ ]:
```

1.2.2 Part (b)

[10 points]

Consider again a relation $R(X, Y, Z)$ with some list of functional dependencies $f_1, f_2, \ldots, f_n$. Now suppose that K is a **key** for this relation, given these functional dependencies.

CS145 student Bryan claims that if we add any new functional dependency $f_{n+1} = U \rightarrow V$ (where U and V are again subsets of $\{X, Y, Z\}$) to our list of functional dependencies, then K will still be a key for R given $f_1, f_2, \ldots, f_{n+1}$. Is Bryan correct? If yes, explain why. If no, provide a counter-example.

In [ ]:

1.2.3 Part (c)

[10 points]

Consider now a more general relation $R(X_1, X_2, \ldots, X_m)$ with some list of functional dependencies $f_1, f_2, \ldots, f_n$. Suppose again that K is a **key** for this relation, given these functional dependencies (recall that any superkey K must be a subset of the attributes of R, which are $\{X_1, X_2, \ldots X_m\}$).

CS145 student Chris claims that if we add any new functional dependency $f_{n+1} = U \rightarrow V$ (where U and V are subsets of $\{X_1, X_2, \ldots, X_m\}$) to our list of functional dependencies, and the new functional dependency **does not include any of the attributes that are in the key** K (that is, $U \cap K = V \cap K = \emptyset$), then K will still be a key for R given $f_1, f_2, \ldots, f_{n+1}$. Is Chris correct? If yes, explain why. If no, provide a counter-example.

In [ ]:

1.3 Problem 3

[30 points total]

Dan wants to find the keys of a relation based on an instance (the concrete rows stored in a database), rather than the functional dependencies (which are external constraints on the schema), but he isn't sure when this problem is possible. In this problem, you will study potential errors that can occur when using an instance instead of the functional dependencies.

Dan defines a set K to be a **plausible key** for an instance `T` if K could be a superkey for `T`, and no subset of K could possibly be a superkey for `T`. (Equivalently, K is a plausible key for `T` if there is an set of functional dependencies that are consistent with `T` in which K is a superkey, and for all subsets $U \subset K$, there are no sets of functional dependencies that are consistent with `T` in which U is a superkey.)

Consider a relation $R(A, B, C, D)$ that satisfies the following set of functional dependencies:

$$\{A, B\} \rightarrow \{C\} \tag{1}$$
$$\{C\} \rightarrow \{D\} \tag{2}$$
$$\{D\} \rightarrow \{A, B\} \tag{3}$$

1.3.1 Part (a)

[10 points]

Create an instance `T` of R for which $\{A, B\}$ is a plausible key. If you believe that `T` cannot be created, provide it as an empty table.

Use a series of `INSERT` statements below to populate the table `T`.

```
In [ ]: %%sql
        DROP TABLE IF EXISTS T;
        CREATE TABLE T(A int, B int, C int, D int);
```

1.3.2 Part (b)

[10 points]

Create an instance `T` of R for which $\{A\}$ is a plausible key. If you believe that `T` cannot be created, provide it as an empty table.

Use a series of `INSERT` statements below to populate the table `T`.

```
In [ ]: %%sql
        DROP TABLE IF EXISTS T;
        CREATE TABLE T(A int, B int, C int, D int);
```

1.3.3 Part (c)

[10 points]

Create an instance `T` of R for which $\{A, B, C\}$ is a plausible key. If you believe that `T` cannot be created, provide it as an empty table.

Use a series of `INSERT` statements below to populate the table `T`.

```
In [ ]: %%sql
        DROP TABLE IF EXISTS T;
        CREATE TABLE T(A int, B int, C int, D int);
```

1.4 Problem 4

[20 points total]

1.4.1 Part (a)

[10 points]

Consider a schema $R(A, B, C, D)$ which has functional dependencies

$$\{A, C\} \to \{B\} \tag{4}$$
$$\{D\} \to \{C\}. \tag{5}$$
$$\tag{6}$$

Create an instance of R which is consistent with these functional dependencies, but not with any other functional dependencies (i.e. **all functional dependencies that are not inferred from these are violated**).

Use a series of `INSERT` statements below to populate the table `R`.

```
In [ ]: %%sql
```

1.4.2 Part (b)

[10 points]

Now consider a schema $S(A, B, C, D, E)$ which has an additional attribute E, but the same functional dependencies

$$\{A, C\} \to \{B\} \tag{7}$$
$$\{D\} \to \{C\}. \tag{8}$$
$$\tag{9}$$

Create an instance of S which is consistent with these functional dependencies, but not with any other functional dependencies (i.e. **all functional dependencies that are not inferred from these are violated**).

Write one or more SQL statements that populate the table `S` using the table `R` you constructed in part (a). You may find the following SQL query helpful: **INSERT INTO S SELECT [SELECT CLAUSE] FROM R;**, where **[SELECT CLAUSE]** is something you will have to fill out.

```
In [ ]: %%sql
        DROP TABLE IF EXISTS S;
        CREATE TABLE S (A int, B int, C int, D int, E int);
```