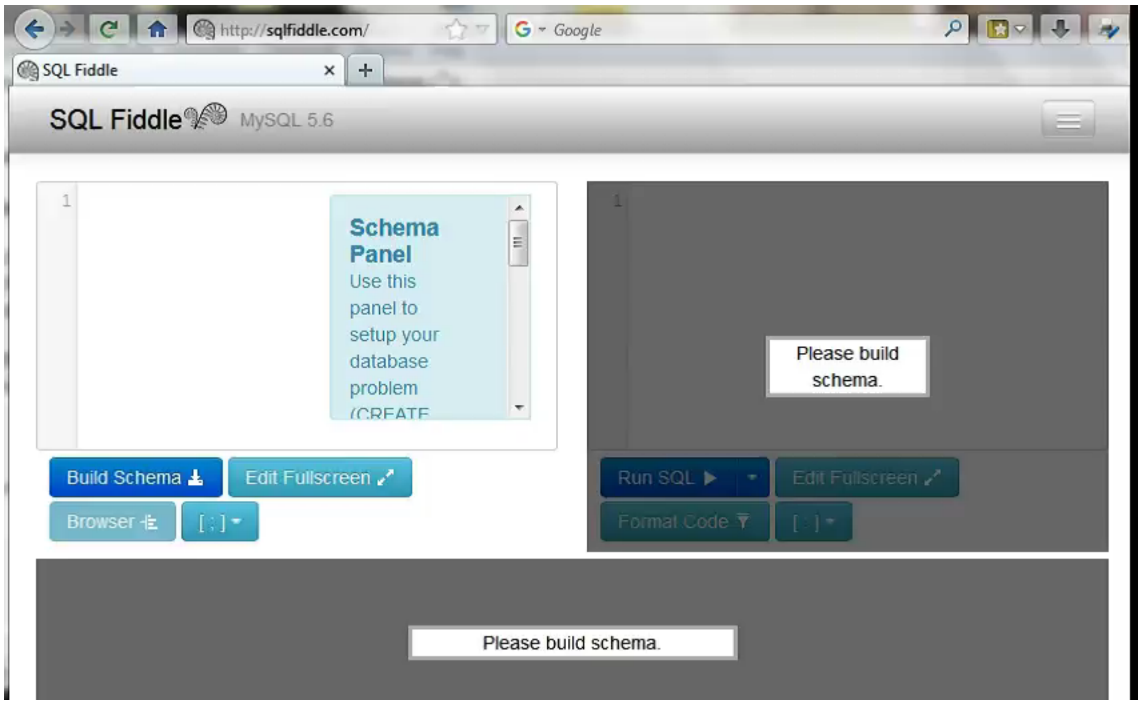
HW1: 5*4=20 (+2 bonus) points : assigned 5/27, due before 11:59PM on 6/3

In this assignment, you will code solutions to **five** SQL problems given below.

You need to use sqlfiddle (sqlfiddle (http://www.sqlfiddle.com (http://www.sqlfiddle.com)) to do your homework.

In sqlfiddle, you create the schema (create table, insert rows) on the left pane, and run queries on the right pane (results are printed in the bottom pane):



00:00 -00:24

Each successful change you make (when you edit and re-run) results in a new, unique URL, which 'persists' (the session is saved to disk at sqlfiddle.com; you (or anyone you send the link to) can re-visit the URL at a future date, at which point your session would be restored). sqlfiddle.com does keep users' contents (in those URLs) for a very long time (months); but, just for safety, please do make a local backup copy on your computer, of the SQL code you submit, in case your links become inaccessible. Note - if your sqlfiddle content becomes unavailable for any reason, so will everyone else's, including mine!. Translation: if the "dog ate your homework", it would have eaten all of ours (so it is not going to be an easy excuse for why you are late in submitting).

ALL the SQL knowledge/commands you need to answer the questions have been covered in class! You do NOT need to learn more commands or techniques (eg. use of 'triggers') etc. on your own in order to do this HW set.

Please submit your answers in the form of five (or six or seven!) sqlfiddle URLs [one for each of the problems below] in a single README file, eg.

```
Q1: http://sqlfiddle.com/#!9/36b044
Q2: ...
...
Q5: ...
[Q5 bonus #1:...]
[Q5 bonus #2:...]
```

Your grader will visit each URL, examine your code and grade your work. It is your responsibility to submit valid links.

You need to turn in your work, using the Dropbox submission folder in D2L (look in MY TOOLS -> Dropbox).

You can post HW1-related questions on D2L, in the HW1 forum (MY TOOLS -> Discussions -> HW1 Forum).

Q1 (use PostgreSQL 9.3). Here is a table for recording guests' stays at a hotel (arrDate denotes arrival date, depDate is departure date):

```
CREATE TABLE HotelStays
(roomNum INTEGER NOT NULL,
arrDate DATE NOT NULL,
depDate DATE NOT NULL,
guestName CHAR(30) NOT NULL,
PRIMARY KEY (roomNum, arrDate));
```

There are two problems (issues) with the above. First, the arrival date could be incorrectly entered to be later than the departure date. Second, a new entry (for a new guest) could be put in for a room number, even before the existing guest has checked out:

```
INSERT INTO HotelStays(roomNum, arrDate, depDate, guestName)
VALUES
(123, to_date('20160202', 'YYYYMMDD'), to_date('20160206','YYYYMMDD'), 'A'),
(123, to_date('20160204', 'YYYYMMDD'), to_date('20160208','YYYYMMDD'), 'B'),
(201, to_date('20160210', 'YYYYMMDD'), to_date('20160206','YYYYMMDD'), 'C')
;
```

How would you redesign the table to fix both these issues? Hint - "do not be concerned with efficiency" - ANY working solution is acceptable :)

Q2. Given the following enrollment table, **write a query to create a view** that includes course name and the number of students enrolled in the course.

```
SID  ClassName Grade

123  ART123    A
123  BUS456    B
666  REL100    D
666  ECO966    A
666  BUS456    B
345  BUS456    A
345  ECO966    F
```

Given the above, the output needs to be:

```
ClassName  Total

ART123     1
REL100     1
ECO966     2
BUS456     3
```

Q3. Below is a small table that tracks work orders for projects. We have a project ID column on the left, a 'step' column in the middle (0,1,2.. denote steps of the project), and a status column on the right ('W' denotes 'waiting', 'C' denotes 'completed'). So the table lets project managers get a quick status on the various aspects (steps) of their projects ("where they're at", in colloquial, ungrammatical language).

```
ProjectID  Step  Status

P100       0     C
P100       1     W
P100       2     W
P201       0     C
P201       1     C
P333       0     W
P333       1     W
P333       2     W
P333       3     W
```

Write a query to output the project(s) where step 0 has been completed, ie. the project gotten started but the rest of the steps are in waiting mode. In the above table, this would output just P100.

Q4. Below is a small sample of a junkmail database, ie. people we want to spam via postal mail.

```
Name      Address  ID  SameFam

'Alice'   'A'      10  NULL
'Bob'     'B'      15  NULL
'Carmen'  'C'      22  NULL
'Diego'   'A'      9   10
'Ella'    'B'      3   15
'Farkhad' 'D'      11  NULL
```

Each entry consists of a name, address, ID, and whether there is a prior family member already in the db; for the last column, NULL means that entry is the first in the family, and a non-null value is the ID of the first family member (eg. Diego points to Alice, and Ella points to Bob).

Write a query to delete from the table, names that have 'NULL' for SameFam and have other family members in the db. So in our example above, the query would delete Alice and Bob, but not Carmen and Farkhad.

Q5. Below is a table of chefs and what they know to make:

```
Chef         Dish

'A'          'Mint chocolate brownie'
'B'          'Upside down pineapple cake'
'B'          'Creme brulee'
'B'          'Mint chocolate brownie'
'C'          'Upside down pineapple cake'
'C'          'Creme brulee'
'D'          'Apple pie'
'D'          'Upside down pineapple cake'
'D'          'Creme brulee'
'E'          'Apple pie'
'E'          'Upside down pineapple cake'
'E'          'Creme brulee'
'E'          'Bananas Foster'
```

Write a query that will pick out all those chefs who can make every dish in the table below:

```
'Apple pie'
'Upside down pineapple cake'
'Creme brulee'
```

With the above data, the query would output

```
Chef

'D'
'E'
```

You will get 1 extra point if you can reformulate the query in a very different way! If you do this, make it a separate fiddle, so you'd submit 6 URLs total. If you come up with YET ANOTHER way, you can get 1 *more* bonus point (please submit 7 URLs in this case).