HW1: assigned 9/17, due 10/1 before 11:59 PM

Total points: 5+5=10

In this assignment, you will code solutions to the pair of SQL problems given below.

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.

You can use any of the ways mentioned in <https://dl.dropboxusercontent.com/u/91263185/!shared/courses/CS585/f16_DSRDB/lectures/SQL/slides.html#%2814%29>(a remote DB via an online shell, a locally installed DB or a cloud DB) to do the problems. What you need to submit is a text file with the SQL commands that you come up with, one for each question (Q1.sql, Q2.sql). Your grader(s) will execute the commands you submit, to see if they produce the expected results.

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

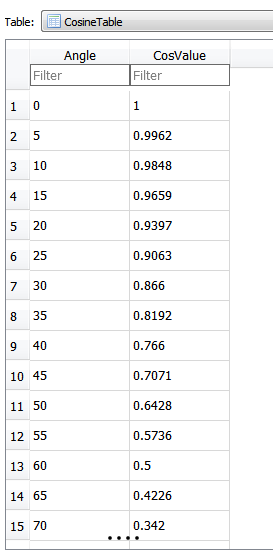
You can post HW1-related questions on D2L, in the HW1 forum (MY TOOLS -> Discussions -> HW1 Forum).

Please see your TAs/graders if you need one-on-one help (or see me or Olivera). Good luck, have fun!

Q1. Cosine table and interpolation (2+3=5 points)

In this question, we're going to use some trig and do some "lerp" :)

1. Write SQL statements to create a table called 'CosineTable', a part of which is shown below:



The table needs to contain (just) these values, for the Angle and CosValue columns:

(0,1),

(5,0.9962),

(10,0.9848),

(15,0.9659),

(20,0.9397),

(25,0.9063),

(30,0.866),

(35,0.8192),

(40,0.766),

(45,0.7071),

(50,0.6428),

(55,0.5736),

(60,0.5),

(65,0.4226),

(70,0.342),

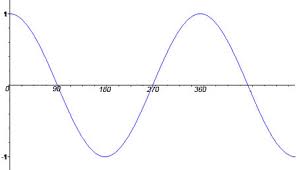
(75,0.2588),

(80,0.1736),

(85,0.0872),

(90,0)

As you can see, we're storing a cos() table, where the angle is expressed in degrees (not radians), in increments of 5, from 0 to 90 degrees. Using these we can derive values for other angles such as 120 degrees, 225 degrees etc. (when they are multiples of 5) by 'mirroring' and 'flipping' the 0-90 curve/table. Table lookups are faster than using native calls (eg. cos(), sin(), tan()..), that is the rationale for such an approach.



2. Next, we want to do "lerp", which is short for 'linear interpolation'. In other words, what if we want the value for cos(73) [73 degrees], given the values in the above table? Rather than compute cos(73) mathematically (eg. using a series formula) or using a builtin (eg. vendor-supplied) call, imagine that we are required to make use of the values in our stored table for the computation - reason being, this would be faster.

As for cos(73), our table contains values in 5 degree increments, so we'll need to use the two 'bracketing' values cos(70) and cos(75), to find the result via lerp:

Calculate cos(73) using table lookups, like so:

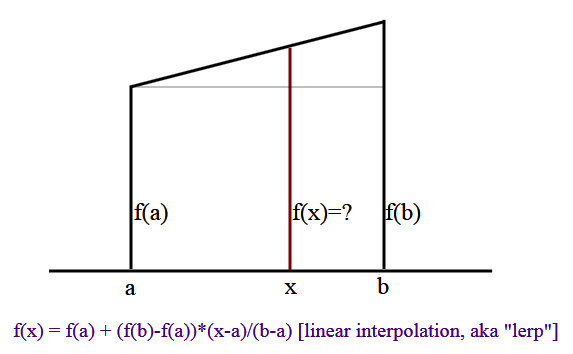
cos(73) = cos(70) + (cos(75)-cos(70))\*(73-70)/(75-70) = 0.342 + (0.2588-0.342)\*(3/5) = 0.342+ -0.04992 = 0.29208

The above isn't too bad, if we need just 2 or 3 digit accuracy, since cos(73) is [0.29237170472](https://www.google.com/search?q=cos%2873*pi%2F180%29&ie=utf-8&oe=utf-8).

lerp is expressed this way: given two values a and b (endpoints) and a value x inside the a to b range, and given f(a) and f(b):

lerp: f(x) = f(a) + (f(b)-f(a)) \* (x-a)/(b-a)

Graphically:



In our example above, a=70, x=73, b=75, f(a)=cos(70), f(b)=cos(75). As a quick check, if you substitute x=a in the lerp formula, you'll get f(a), and when x=b, you'll get f(b) - as you should expect.

A table like the above might be stored in the ROM of a cheap quadcopter, for example, where high precision is not needed for its operation - turning angles can be computed by looking up one of 19 values (in our example) from a LUT (lookup table), which can be a simple array.

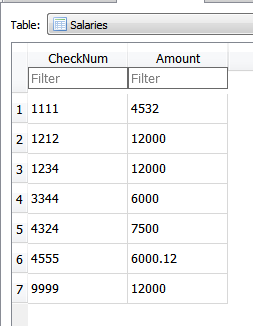
You need to write a query (which would have the expected keywords SELECT, FROM, WHERE) that has a hardcoded angle in it, eg. 43, 12, 88, 62 etc. - in each case, it needs do a lerp and print out a single value as the output, which would be the interpolated cosine value. Eg. embedding 73 in your query should print out 0.29208 (see the example above). HINT: your SQL query would have the angle value hardwired in it (eg. 73) in it, THREE times.

Note: your query needs to just handle angles in the 0..90 degree range, no need for 'flipping and mirroring'; also, your query does not need to handle multiples of 5 angles, ie. 0,5,10..

What to submit: a Q1.sql text file, with the commands from above - for table creation, and doing the lerp(). Also, BE SURE to mention which DB or URL you used (this is so the grader can attempt to reproduce your results)!

Q2. Descriptive statistics (5 points)

You are given a table of salaries, like so:



The above table has just 7 entries, in the real world a similar table might have 700, or even 70,000 entries. The task is to find the amount that occurs the most # of times, and also the count (how many times). With the above data, these would be 12000,3. Interestingly, SQL does not come with a built-in function for this common operation :)

Write a SQL query that outputs the two values mentioned above (most occuring amount, and the count). Needless to say, your query should work on ANY such table (ie. you can't hardcode 12000,3 in your query!!!).

You can assume the following table structure (feel free to fill it with the data shown above, or put in your own):

CREATE TABLE Salaries(

CheckNum INTEGER PRIMARY KEY,

Amount DECIMAL(8,2) NOT NULL

);

Note - if there are multiple amounts that occur the most # of times (eg. if 8500 also occurs thrice in the above sample table), your query needs to output all of such amounts, not just one.

What to submit: a Q2.sql text file, with your query in it.

To reiterate, everything you need is in the slides, of the material we went through in class - you do not need to read ahead! Look at each command, each function, each keyword, and ask yourself how it could be of use in constructing your query. Good luck, have fun figuring it all out!!