Assignment 2 – MySql (Very Easy!)

It is that time of year again. The ITC class has ended at UCP and the professor has to submit the final grades. The school requires grades to be submitted with a letter grade. In addition the grades should be submitted from the "best" student first. The individual scores should be be listed from "worse" to "best".

Design a Table in MySql with the following attributes..

(first name) , (last name) (score 1) (score 2) (score 3) (score 4) (score 5)

Each student has 5 scores from 1 to 100. So a total of 500 points for the class. Based on this you must determine what grade they get on a percentile 1-100.

The letter Grades are assigned based on the following

* 90-100 A
* 80-89 B
* 70-79 C
* 60-69 D
* 59 and below F

Those scoring in the top 3 percent of the rank get a "+" added. Those scoring in the lower 3 percent of the rank get a "-". However there is no "+" for an A and there are no "+" or "-" for an F grade.

student scoring 82% would be a B-

student scoring 79% would be a C+

Note: Final Grades are rounded to the nearest whole number. So 89.5 is 90 and 89.4 is 89.

**Output:**

The output should be ranked from the "best" student who had the best grade to the "worse" student who had the lowest grade. The 5 scores should also be arranged from the "lowest" to "highest".

The output should take on this form:

(Last Name) (First Name) (Final percentage) (Final Grade) : (Scores 1-5 from low to high)

Example:

Valerie Vetter 79 81 78 83 80

Richie Rich 88 90 87 91 86

Would output as:

Rich Richie (88%) (B+): 86 87 88 90 91

Valerie Vetter (80%) (B-): 78 79 80 81 83

# Formatting:

Optional but keep in mind the dean of the college is going to be looking at this and it might be fun and a good idea to align the output to make it look nice and easy to read.

# Challenge input:

Jennifer , Adams 100 70 85 86 79

Bubba , Bo Bob 50 55 60 53 30

Matt , Brown 72 82 92 88 79

Ned , Bundy 73 75 80 79 88

Alfred , Butler 80 90 70 100 60

Sarah , Cortez 90 72 61 70 80

William , Fence 88 86 83 70 79

Casper , Ghost 80 85 87 89 90

Opie , Griffith 90 90 90 90 90

Tony , Hawk 60 60 60 72 72

Kirstin , Hill 100 90 92 94 95

Hodor , Hodor 40 50 53 62 33

Clark , Kent 89 90 88 92 91

Tyrion , Lannister 93 97 100 91 95

Ken , Larson 70 80 85 73 79

Stannis , Mannis 60 70 75 77 78

Bob , Martinez 79 88 92 82 72

Jean Luc , Picard 90 89 95 70 65

Harry , Potter 73 75 77 69 73

Jaina , Proudmoore 90 92 100 95 94

Richie , Rich 88 90 87 91 86

John , Smith 90 80 70 60 50

Jon , Snow 70 70 70 70 72

Arya , Stark 91 92 90 93 90

Edwin , Van Clef 40 50 55 57 33

Valerie , Vetter 79 81 78 83 80

Katelyn , Weekes 90 95 92 93 97

Wil , Wheaton 70 80 75 71 77

Steve , Wozniak 88 89 87 86 85

Derek , Zoolander 80 81 85 88 90

You may output the query result to HTML if you wish to integrate. However, it is not necessary – So, all you have to do is to design the correct table and write the appropriate queries so that when we run the query the data is fetched from db in the expected format..

Solution (Please don’t see it till you solve the assignment – its in white color right now to hide the solution..

http://www.reddit.com/r/dailyprogrammer/comments/28gq9b/6182014\_challenge\_167\_intermediate\_final\_grades/#

SQL, with a bit of Perl to massage the input. I'll be using SQLite's flavor of SQL to generate the report.

I created two tables with a many-to-one relationship to model the data.

* students: lists all of the students *without* their scores
* scores: lists all of the scores with a foreign key on students

For simplicity, I'm assuming last names are unique so that I can use them as a primary key.

-- Table of all students

CREATE TABLE students (last PRIMARY KEY, first);

-- Table of all scores for all students

CREATE TABLE scores (last REFERENCES students(last), score);

I also create a totals *view* on a [JOIN](http://en.wikipedia.org/wiki/Join_%28SQL%29) between these two tables. Joins are the bread and butter of relational databases. This view will look and feel just like a normal table, and it will always be up to date. Later on this will make queries involving averages much simpler because we can join with this view.

-- Create a running averages "table"

CREATE VIEW totals AS

SELECT last, CAST(avg(score) AS INTEGER) AS total

FROM students NATURAL JOIN scores

GROUP BY last;

I also create a table grades for letter grades. I can join with this table in my report to get a letter grade column.

CREATE TABLE grades (grade PRIMARY KEY, min, max);

INSERT INTO grades (grade, min, max) VALUES ('A', 93, 101);

INSERT INTO grades (grade, min, max) VALUES ('A-', 90, 93);

INSERT INTO grades (grade, min, max) VALUES ('B+', 87, 90);

INSERT INTO grades (grade, min, max) VALUES ('B', 83, 87);

INSERT INTO grades (grade, min, max) VALUES ('B-', 80, 83);

INSERT INTO grades (grade, min, max) VALUES ('C+', 77, 80);

INSERT INTO grades (grade, min, max) VALUES ('C', 73, 77);

INSERT INTO grades (grade, min, max) VALUES ('C-', 70, 73);

INSERT INTO grades (grade, min, max) VALUES ('D+', 67, 70);

INSERT INTO grades (grade, min, max) VALUES ('D', 63, 67);

INSERT INTO grades (grade, min, max) VALUES ('D-', 60, 63);

INSERT INTO grades (grade, min, max) VALUES ('F', 0, 60);

Before I can make any use of the data, it must be converted into SQL INSERT statements. This Perl script will do that.

#!/usr/bin/env perl

while (<>) {

my ($first, $last, @s) =

m/([a-zA-Z ]+?) +, +([a-zA-Z ]+?) +(\d+) +(\d+) +(\d+) +(\d+) +(\d+)/;

print "INSERT INTO students (first, last) VALUES ('$first', '$last');\n";

for (my $i = 0; $i < 5; $i++) {

print "INSERT INTO scores (last, score) VALUES ('$last', $s[$i]);\n";

}

}

The output looks like this,

INSERT INTO students (first, last) VALUES ('Jennifer', 'Adams');

INSERT INTO scores (last, score) VALUES ('Adams', 100);

INSERT INTO scores (last, score) VALUES ('Adams', 70);

INSERT INTO scores (last, score) VALUES ('Adams', 85);

INSERT INTO scores (last, score) VALUES ('Adams', 86);

INSERT INTO scores (last, score) VALUES ('Adams', 79);

INSERT INTO students (first, last) VALUES ('Bubba', 'Bo Bob');

INSERT INTO scores (last, score) VALUES ('Bo Bob', 50);

-- ...

The tables are in place so now I can generate a report. This involves joining all of the tables above. A *natural* join means that when the column names match (last in this case) we can join wherever they hold equivalent values. To get a letter grade, I also join on the grades table using <= and > operators to join on the correct grade letter.

SELECT first, last, total, grade

FROM totals

NATURAL JOIN students

JOIN grades ON total >= min AND total < max

ORDER BY total DESC, last ASC;

The output:

Tyrion Lannister 95 A

Kirstin Hill 94 A

Jaina Proudmoore 94 A

Katelyn Weekes 93 A

Arya Stark 91 A-

Opie Griffith 90 A-

Clark Kent 90 A-

Richie Rich 88 B+

Steve Wozniak 87 B+

Casper Ghost 86 B

Jennifer Adams 84 B

Derek Zoolander 84 B

Matt Brown 82 B-

Bob Martinez 82 B-

William Fence 81 B-

Jean Luc Picard 81 B-

Alfred Butler 80 B-

Valerie Vetter 80 B-

Ned Bundy 79 C+

Ken Larson 77 C+

Sarah Cortez 74 C

Wil Wheaton 74 C

Harry Potter 73 C

Stannis Mannis 72 C-

John Smith 70 C-

Jon Snow 70 C-

Tony Hawk 64 D

Bubba Bo Bob 49 F

Hodor Hodor 47 F

Edwin Van Clef 47 F

Unfortunately I'm still too much of a SQL newb to figure out how to list all of the scores in order in additional columns. There's probably some trick involving five left outer joins or something. I'm still working on that part.

This may seem a little bulky for such a small report, but it would scale up enormously well. After adding a few indexes in the right places, you could do all sorts of concurrent, fast queries in involving millions of students with thousands of scores each while safely making updates to the database at the same time.