CS383 Midterm

Name:

Start Time:

Finish Time:

Accommodation (if applicable):

I have abided by the Honor Code. I have not discussed this test with anyone.
(Sign below)

You have 100 minutes from the time you downloaded this test until you return the completed test.

If you take this test on separate sheets of paper, put all of the items above on your first page. If you need more space, feel free to add extra pages. Just make sure everything is well labelled.

All SQL Questions use the univ database (unless explicitly stated otherwise).

There are 7 questions on 12 pages in this test. As indicated in the table below some questions have options (do either A or B). Do not answer both A and B. Also, some questions have 2 parts. Be sure to answer both parts of those questions. Especially in the SQL questions, if you cannot devise answer the entire question, devise a partial answer and explain (1 sentence) why your partial answer is on the part of a solution.

Question	**parts**	**points**
1	A or B	10
2	A or B	10
3	A or B	10
4	A (2 parts) or B (2 parts)	13
5	2 parts	13
6	A (2 parts) or B	20
7		20
Possible Points:		96

Question 1 (10 points).

Write an SQL query for either A or B

A. Find all instructors and their departments such that the name of the instructor has at least 2 instances of the character ‘a’ and the department name has at least one instance of the character ’e’. Do not worry about capitalization.

```
univ=# select name, dept_name from instructor where name like '%a%a%' and dept_name
like '%e%';
  name  | dept_name
--------+-----------
 Murata | Athletics
 Bawa   | Athletics
(2 rows)
```

B. Find the names of all students who have taken the fewest of credits of any student who has taken more than 0 credits.. (Do not hard code the number of credits into the query.)

```
univ=# select name, tot_cred from student where tot_cred=(select min(tot_cred) from student
where tot_cred>0);
    name    | tot_cred
------------+----------
 Karlsson   |        1
 Cordt      |        1
 Drews      |        1
 Cheah      |        1
 Tyler      |        1
 Lesaffre   |        1
 Kashima    |        1
 Mitsuhashi |        1
```

Question 2 (10 points).

Write an SQL query for either A or B

A. Find the number of students that have the same name as some other student. Your query must return this number and only this number.

```
select count(studenta.name) from student as
studenta join student as studentb on
studenta.name=studentb.name and studenta.id!
=studentb.id;
1042

with multnames as (with namecounts as (select
name, count(*) as count from student group by
name) select name, count from namecounts where
count > 1) select sum(count) from multnames;
788
– 7 of 10

select count(name)
from student as t1
where t1.name = ANY (select name
from student as t2
where t1.id != t2.id);
788
-7 of 10
```

B. Find all courses such that they have a time_slot that is not defined in the time_slot table.

```
select distinct course_id from section where
time_slot_id not in (select time_slot_id from
time_slot);
```

Question 3 (10 points).

Write an SQL query for either A or B

A. Generate a table that shows, for each advisor who advises less than 30 students, the name of each of their advisees. You may have columns in your result in addition to the advisor and adivisee names.

```
with aa as (select i_id, count(*) from advisor
group by i_id having count(*)<30), bb as (select
aa.i_id, s_id from advisor, aa where
aa.i_id=advisor.i_id) select * from bb, student,
instructor where bb.i_id=instructor.id and
bb.s_id=student.id;
```

B. For each department, how much money is left over after taking out the salaries of all instructors?

```
with aa as (select sum(salary) as ss, dept_name
from instructor group by dept_name) select
aa.dept_name, budget-ss from aa, department
where aa.dept_name=department.dept_name union
(select dept_name , budget from department where
dept_name not in (select distinct dept_name from
instructor));
```

```
  dept_name  |  ?column?
-------------+------------
 Psychology  |  725888.94
 Geology     |  307175.34
 Civil Eng.  |  255041.46
 Mech. Eng.  |  360724.61
 Biology     |  525036.05
 Math        |  777605.11
 Marketing   | -125762.17
 Cybernetics |  409155.19
 English     |  322686.46
```

Pol. Sci.	273585.87
Finance	761520.37
Astronomy	538183.86
Statistics	-11720.91
Physics	713008.96
Comp. Sci.	-89888.25
Accounting	246974.55
History	699140.86
Languages	429018.03
Elec. Eng.	-20123.35
Athletics	349059.71

Question 4 (13 points).

Write SQL for either A or B

A: You suspect two students of cheating because they always seem to take the same section of the same course in the same semester. For this question you may assume that the students have ids '336' and '94257'

Part A1. (10 points) Write a query using a set operator (union, etc) that returns a list of the course, section, year and semester of all courses that the two students have taken together.

Part A2. (3 points) Rewrite the query to show the courses that student 336 has taken without student 94257.

```
(select course_id, sec_id, semester, year from
takes where id='336') intersect (select
course_id, sec_id, semester, year from takes
where id='94257');

(select course_id, sec_id, semester, year from
takes where id='336') except (select course_id,
sec_id, semester, year from takes where
id='94257');
```

B: You suspect that the quality of teaching varies with instructor salary. To investigate this:

Part B1: (10 points) Write an SQL function that returns the average of the lowest and highest instructor salaries. (The answer to part B1 should start: "create or replace function …")

Part B2: (3 points) Use the function from part B1 in the following query: Get a listing of all courses taught by instructors whose salary is less than the average of the highest and lowest instructor salaries. (For this part you do not need a working version of the part B1 function, just write this query assuming you have a working version of the function.)

```
create or replace function avvg() returns float
as $$ with aaa as (select min(salary) as minn
from instructor), bb as (select max(salary) as
maxx from instructor) select (aaa.minn +
bb.maxx)/2 from aaa, bb $$ language SQL;

select * from instructor where salary>avvg();
```

Question 5 (13 points).

Part 1. (8 points) The HAVING clause of SQL is little used. So, write the following query using HAVING in an appropriate way. In the student table, assume that dept_name indicates that student's major. Use the HAVING clause to create a list department and student major count of all departments that have more than 100 students in the major.

```
select count(*), dept_name from student group by
dept_name having count(*)>100;
```

Part 2: (5 points) The HAVING clause is is fairly unique among the standard part of SQL in that it is not actually necessary. Rewrite the query from Part 1 so that it returns exactly the same results without using HAVING. (It is possible to not do part 1 and get full credit on part 2).

```
with aa as (select count(*) as cc, dept_name
from student group by dept_name) select * from
aa where cc>100;
```

Question 6 (20 points).

Do either A or B (put your answer on the next page)

A1: (15 points) Write an entire web page with included javascript to do the following.

- There should be an html element that starts a javascript method when the element is clicked upon.
- The javascript function that is started should do two things:
  - change the text of the clicked element to the text “clicked 1”
  - start a timer that completes in 5000ms
  - When the timer completes change the text of the html element to “clunked 1”
  - When the html element is hit a second time (after the completion of the timer), change the text to “clicked 2” and start the timer which, on completion (after 5000ms) changes the text to “clunked 2”.
  - “clicked 3” and “clunked 3”
  - ect.

Part A2: Suppose the viewer of the page clicked on the html element a second time before the 5000ms timer competes (say after 4700ms). What happens? (What does the viewer of the page see?) Explain. (This part supposes that you did nothing in part A1 to block the effect of clicking on the html element while the timer is running. Hence, an answer of “nothing” is not acceptable)

```
<html>
    <body>
        <button onclick="clickk()" id="gt">Button</button>
        <script>
            let cnt=0;
            function clickk() {
                cnt++;
                document.querySelector("#gt").innerHTML = `clicked $
{cnt}`;
                setTimeout(function() {
                    document.querySelector("#gt").innerHTML = `clunked
${cnt}`;
                }, 5000);
            }
        </script>
    </body>
</html>
```

If you hit the button a second time, that will start a second independent timer after incrementing cnt. As a result, when the first times completes it will change the text in the button to “clunked 2”. When the second timer

completes, it too will change the text, but the change will be invisible as the tet will be changed from “clunked 2” to “clunked 2”.

B: With precision, explain what happens and why when you click on the Button1, Button2, Button1, Button1, Button2, Button1, Button1, … in the following web page. (This page is available at https://cs.brynmawr.edu/cs383/TEST/test1.html)

```
<html>
    <body>
        <button onclick="kk()">Button1</button>
        <button onclick="jj(3)">Button2</button>
        <script>
            let kk = null;
            let hh=1;
            function jj(param1) {
                let zz = hh + param1;
                kk = function() {
                    console.log(`thatt ${zz++}`);
                }
                zz*=10;
                hh*=2;
            }
        </script>
    </body>
</html>
```

The script is initially executed in order setting kk to null, hh to 1, function jj() is defined but not executed. kk is defined inside jj() and jj() has not been executed, so the first time Button 1 is pushed, it tries to call kk() but since it hasn’t been defined yet there is an Uncaught TypeError: kk is not a function. Then Button 2 is pushed and jj() is executed, so zz becomes 1+3=4, then kk() is defined but not executed, then zz becomes 4*10=40 and h becomes 1*2=2. Then Button 1 is pressed and kk() is executed, printing zz’s current value (40) then incrementing zz. Button 1 is pressed again so kk() is executed, printing 41 and incrementing zz. Then Button 2 is pressed so jj() is executed, setting zz to 2+3=5, then zz to 5*10=50, and hh to 2*2=4. Button 1 is pressed so kk() is executed, printing zz (50) and incrementing zz. Button 1 is pressed again so kk() is executed again, printing 51 and incrementing zz.

Question 7 (20 points).

The following web page and node.js instruction file connects with postgres and retrieves some information from the rocket database when you hit the “Fetch Data” button. The information retrieved is not relevant. There are a lot of asynchronous elements in the web page and node.js file. Identify each and explain why asynchrony is needed. One of two sentences should be sufficient in every case.

Web page: available at: http://165.106.10.170:30045/page.html

```
<html>
    <script language="javascript">
        function doQuery() {
           let params = {
                    method: "POST",
                    headers: { 'Content-type': 'application/json'
                    }}
            let uurl = "http://165.106.10.170:30045/dq1"
            params['body']=JSON.stringify({count:7, userID:3 });
            fetch(uurl, params)
                .then(function(response) {
                    response.text().then(function(text) {
                        let result2 = JSON.parse(text);
                        console.log(result2);
                        document.querySelector("#gt").innerHTML = text;
                    });
                });
        }
    </script>
    <body>
        <br><button onclick="doQuery()" name=“query">Fetch Data</button>
        <div id="gt"></div>
    </body>
</html>
```

```
// Node.js instruction file

const path = require('path')
const express = require('express')
const { Pool } = require('pg') // connecting to postgres
const { CommandCompleteMessage, closeComplete } = require('pg-protocol/dist/
messages')
const pool = new Pool({
    user: 'dbuser',
    host: 'localhost',
    database: 'rocket',
    password: '12345678',
    port: 5432,
})

const app = express()
const port = 30045

app.use("/", express.static(path.join(__dirname)));

function dbreq1(dberr, client, done, req, res) {
    if (dberr) {
        res.writeHead(500);
        res.end('Sorry, check with the site admin for error: ' + dberr.code +
' ..\n');
        return;
    }
    let postt = req.body;
    client.query('SELECT * from vehicle order by random() limit ' +
postt['count'], function (dberr, dbres) {
        done()
        if (dberr) {
            res.writeHead(500);
            res.end('Sorry, check with the site admin for error: ' +
dberr.code + ' ..\n');

        } else {
            res.json(dbres.rows);
        }
    });
};

app.post('/dq1', express.json({ type: '*/*' }), function (req, res) {
    pool.connect(function (dberr, client, done) {
        dbreq1(dberr, client, done, req, res);
    });
});

app.listen(port, function(error){
    if(error) throw error
    console.log(`Server created Successfully on port ${port}`);
})
```

Asynchronous Element	Why it is needed
doQuery	Not exactly async, but rather event-driven. This function wraps the actions that should be taken n response to the click event.
fetch(uurl, params)	Inside the doQuery function, no additional code is executed below this line, as all the future work depends on the result of this fetch operation. This function is not precisely async, but rather it defines things that will be async.
.then(function (response)	The .then() clause after the fetch is necessary so that it can wait for the result of the fetch (callback) to be completed before it starts to work on doing something with the response.
response.text().then (funct ion (text) {	Similarly here, the result of the last anonymous function to be called needs to be completed for us to be able to handle the result. The handling of the result involves parsing JSON. We can't parse the JSON until we've actually gotten it as a response to the query, so we require the .then() construct.
client.query('SELECT * from vehicle order by random() limit ' + posttt['count'], function (dberr, dbres) {	This line runs the query and supplies a function to handle its result. This asynchronous element must be used here in order to prevent node from trying to evaluate the result of the query before it has been returned from the server. The use of an anonymous function to handle the result allows this functionality.
app.listen()	The function inside this gets triggered when the listener setup is complete.
app.post()	This function also has an anonymous function to handle the request and response. As with doQuery above, this is not precisely async so much as it is event driven.
pool.connect()	Used to wait for the DB to admit you. The pool has a limited number of connections to the DB. If there are many users they could all be in use, so you need to wait your turn to talk to the DB.