2024 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 Here)

You have **300** minutes from the time you downloaded this test until you return the completed test.

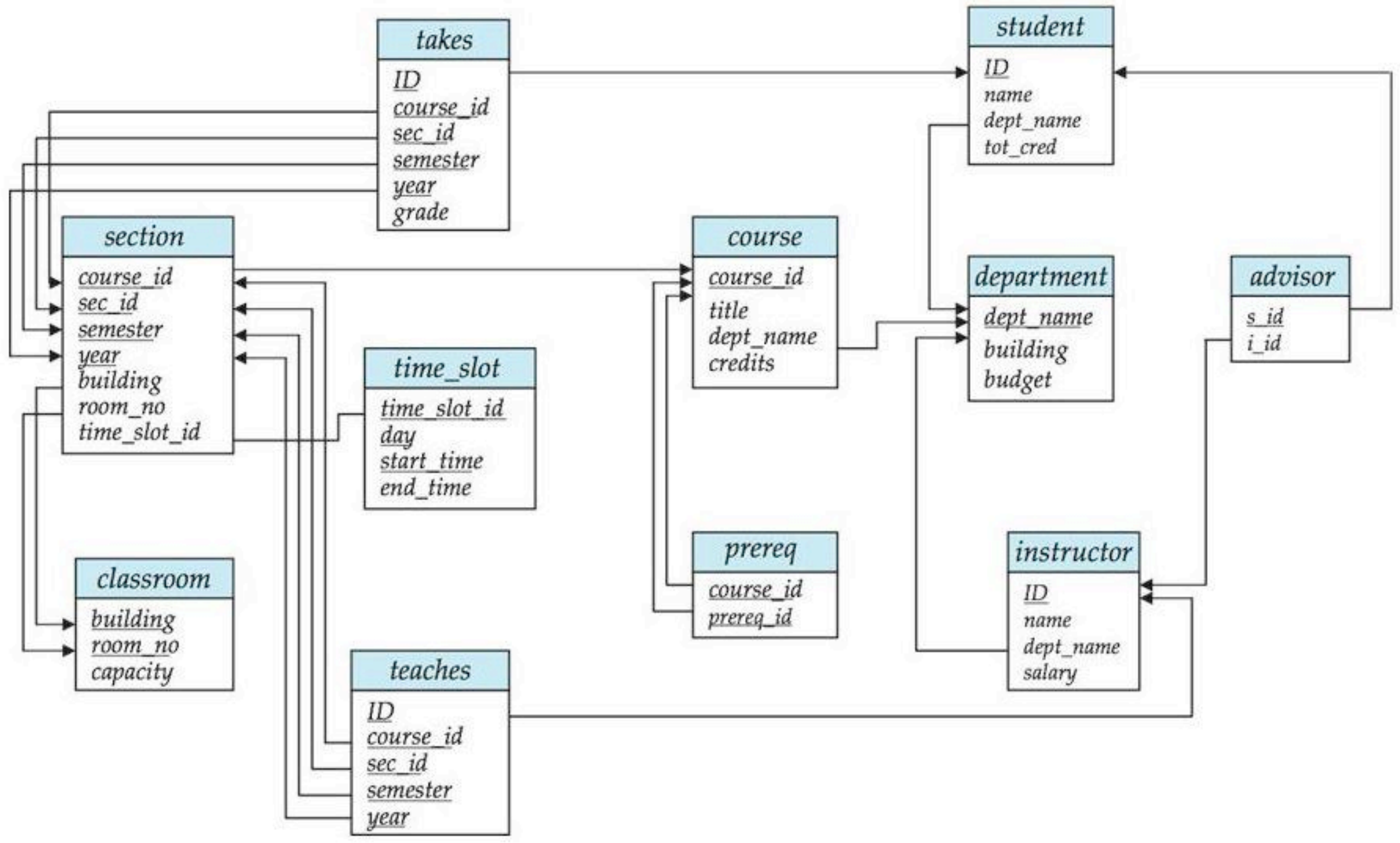
If you take this test on separate sheets of paper, put all of the items above on your first page. If you need more space, feel free to add extra pages. Just make sure everything is well labelled.

The first 2 questions use the univ database which we have previously used. The univ database follows the UML diagram given on the next page.

There are 5 questions on 10 pages in this test (3 of the pages are blank). The first three questions have multiple parts. If you cannot devise answer the entire question, devise a partial answer and explain (briefly) why your partial answer is on the part of a solution.

Question	**parts**	**points**
1	4 (4 each)	16
2	3 (5 each)	15
3	3 (8, 10, 6)	24
4	1	22
5	1	22
Extra Credit (3 points)		
Possible Points:		99

Schema Diagram



Question 1: Single Table Queries on the univ database (4 points each)

A. How many different student names are there?

```
select count(distinct name) from student;
```

B. How many students have a name with at least 3 ‘a’?

```
select count(*) from student where name like
'%a%a%a%';
```

C. What are the highest and lowest student id numbers? (Your query should return a table with exactly one column and exactly two rows.)

```
select max(id) from student union select min(id)
from student;
```

select id from student where id in((select max(id) from student), (select min(id) from student));

D. What are the pre-reqs for the course(s) with the most pre-reqs.

```
with aa(cid, ccount)  as (select course_id,
count(*) from prereq group by course_id) select
* from prereq where course_id in (select cid
from aa where ccount=(select max(ccount) from
aa)) order by course_id;
```

Extra Credit (3 points): List the names of all students whose name has **exactly** 3 ‘a’.

```
select distinct name from student where name
like '%a%a%a%' and name not in (select name from
student where name like '%a%a%a%a%');
```

SELECT name FROM student WHERE LENGTH(name) - LENGTH(REPLACE(name, 'a', '')) = 3;

select * from student where name ~ '^[^a]*a[^a]*a[^a]*a[^a]*$';

Question 2: Multi-Table Queries on the univ database (5 points each)

A. Show the names of all students who have the same name as an instructor

```
select student.name from student join instructor
on instructor.name=student.name;
select student.name from student, instructor
where student.name = instructor.name;
```

B. Find the names of all instructors who have never taught a class

```
select * from instructor where id not in (select
id from teaches);
```

SELECT name FROM instructor LEFT JOIN teaches ON instructor.id = teaches.id WHERE teaches.id IS NULL;

```
SELECT name
FROM instructor
EXCEPT
SELECT instructor.name
FROM instructor
JOIN teaches ON instructor.ID = teaches.ID;
```

C. Show all occurrences of two different students with the same name who received exactly the same grade in a course (they could have taken the course at different times).

```
select * from student as sa join student as sb
on sa.name=sb.name and sa.id<sb.id join takes as
ta on sa.id=ta.id join takes as tb on
tb.id=sb.id and ta.grade=tb.grade and
ta.course_id=tb.course_id;
```

```
SELECT t1.ID AS student_id_1, t1.course_id AS course_id, t1.grade AS
grade, t2.ID AS student_id_2
FROM takes t1
```

```
JOIN takes t2 ON t1.course_id = t2.course_id AND t1.grade = t2.grade

AND t1.ID <> t2.ID
AND t1.ID < t2.ID -- remove duplicate
AND EXISTS (SELECT 1 FROM student WHERE ID = t1.ID AND name
IN

(SELECT name FROM student WHERE ID = t2.ID) -- find same name );
```

Question 3: Normalization

Name1	Name2	Development A1	A1 Firm	A1 Firm HQ	A1 building names	A1Building Sizes	Development A2	A2 Firm	A2 Firm HQ	A2 Building Names	A2 Building Sizes
Frank	Wright	Loud	Def	Spring Green, Wi	[falling, water, shorty*]	[140 ,210, 300]	Tulips and Chimneys	B&M	Chi	[ee, cummings, should, capitalize]	[12, 90, 144, 8128]
Daniel	Burnham	Law	B&M	Chi	[Monadnock, costs]	[200, 300]	Tulips and Chimneys	B&M	Chi	[ee, cummings, should, capitalize]	[12, 90, 144, 8128]

* Burnham contributed to the Wright building “shorty”.

* Note that Wright and Burnham collaborated on the ee cummings themed development “Tulips and Chimneys”.

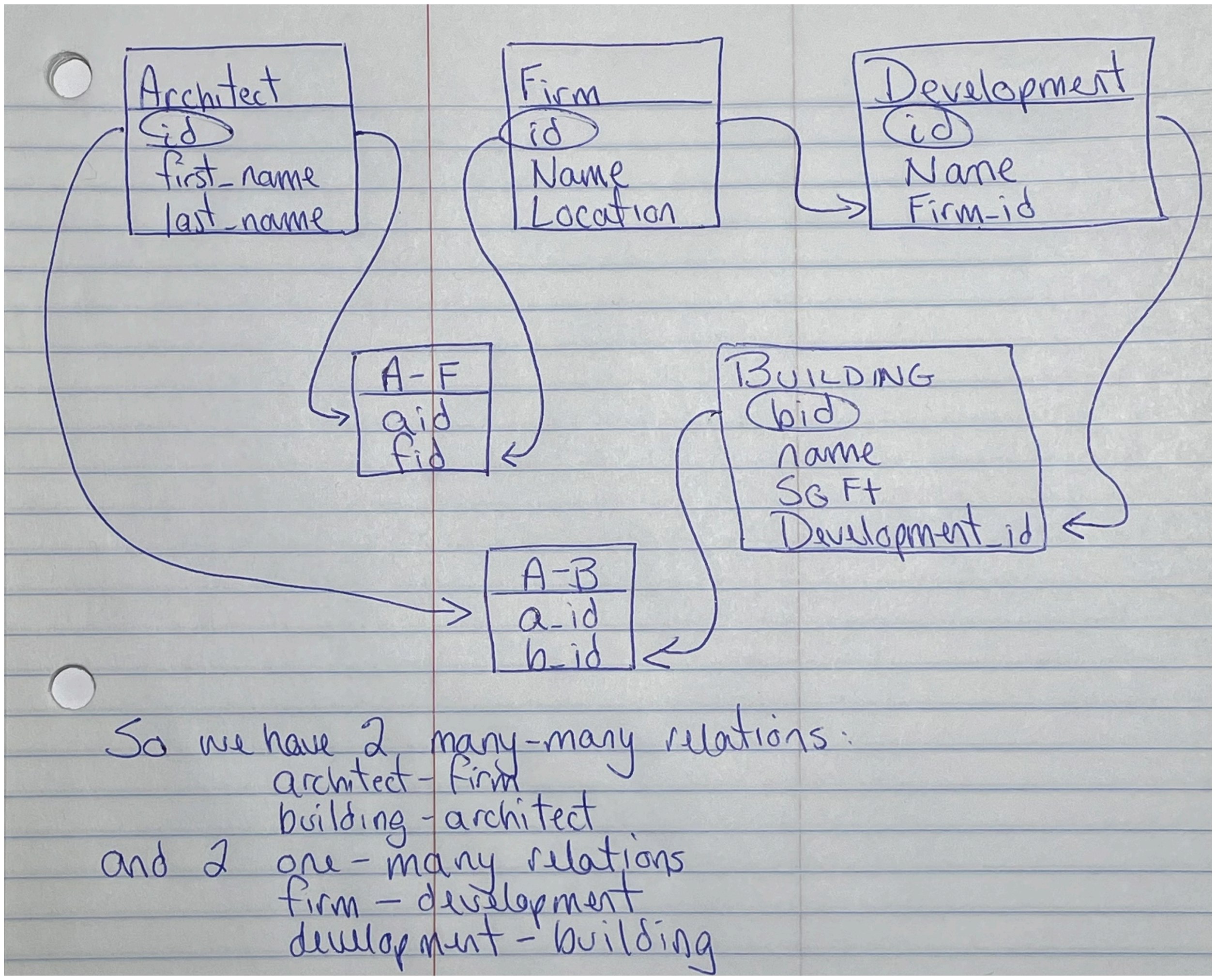
(Daniel Brunham and Frank Loyd Wright never actually worked together.)

A. (8 points) Draw a UML schema diagram (ie something resembling the UNIV diagram on page 2) showing a table design that puts the above data into at least third normal form.

B. (10 points) Write SQL statements to create tables per your Part A drawing. (It is possible to get full credit for this part even if you diagram does not receive full credit.)

C. (6 points) For each of your SQL tables in part B, write one insert statement using above data.

(page intentionally blank)



So we have 2 many-many relations:
architect - firm
building - architect
and 2 one-many relations
firm - development
development - building

Part B

```
create table architect (
    id int generated always as identity,
    first_name varchar,
    last_name varchar,
    primary key (id)
);
create table firm (
    id int generated always as identity,
    name varchar,
    location varchar,
    primary key(id)
);
create table development (
    id int generated always as identity,
    name varchar,
    firm_id int,
    primary key(id),
    foreign key(firm_id) references firm(id)
);
```

```
create table building (
    bid int generated always as identity,
    name varchar,
    sqft int,
    development_id int,
    primary key(bid),
    foreign key(development_id) references development(id)
);
create table a_f (
    aid int,
    fid int,
    foreign key (aid) references architect(id),
    foreign key (fid) references firm(id)
);
create table a_b (
    aid int,
    bid int,
    foreign key (aid) references architect(id),
    foreign key (bid) references building(bid)
);
```

Part C:

```
insert into architect(first_name, last_name) values('frank', 'wright');
insert into firm(name, location) values('Def', 'Spring Green, Wi');
insert into development(name, firm_id) values('Loud', 1);
insert into building(name, sqft, development_id) values('falling water', 140,
1);
insert into a_f values(1,1);
insert into a_b values(1,1);
```

Question 4: (22 points) Javascript & HTML

Write an entire web page with included javascript to do the following.

- The title of the web page should be "Seymour and Audrey 2".
- There should be an html element, not a button, that starts a javascript function when the element is clicked upon.
- The javascript function that is started should do the following:
  - Change the text of the clicked element to "Eating 1"
  - Attempt to get information from the web page "./feed_me.html". (You do not need to write this web page.) This page does not expect any parameters.
  - While waiting for a response, the page should not respond to button clicks
  - Once the web page responds in any way,
    - Change the HTML element to "I have been fed 1"
    - Resume responding to button clicks and do exactly as before except the next time change the element to "Eating 2" and then "I have been fed 2". Etc

(page intentionally blank)

```
<html>
    <head>
        <title>Seymour and Audrey 2</title>
    </head>
    <body>
        <script>
            let ccount = 0;
            let active = true;

            function doQuery() {
                if (!active) {
                    return;
                }
                document.getElementById("#that").innerhtml = `Eating $
{ccount}`;
                ccount ++;
                active=false;
    let params = {
            method: "POST",
        headers: { 'Content-type': 'application/json' },
        body: JSON.stringify({'port':1})
    }
    let uurl = “feed_me.html"
    fetch(uurl, params)
        .then(function(response) {
                active = true;
                document.getElementById("#that").innerhtml = `I have been fed
${ccount}`;

            });
}

        </script>

        <span id="that" onclick="javascript:doQuery()">Click me</span>
    </body>
</html>
```

Question 5: (22 points) Node.js

Write a node.js script that, on receiving a request for “page”, responds with a page that looks like:

```
Hit Count: xx
Fibonacci number: yy
```

Where xx is the number of time this page (and possibly others) have been requested and yy is the xxth Fibonccci number.

For instance, the page returned page might look like:

```
Hit Count: 5
Fibonacci number: 5
```

On the next call the page returned would be:

```
Hit Count: 6
Fibonacci number: 8
```

There exists on the server on which your node.js program is running, a postgreSQL server with a database named ‘hc’ which contains a table named hctable with the following definition

```
create table hctable (
    count int
);
```

This table contains exactly one row, it should never have more than one row. (This is a really stupid database.)

Each time “page” is requested, you must

query postgreSQL for the count,
update the count (increment the count by 1)
return an html page that renders as shown above.

You should not assume that your node.js is the only one accessing hctable, but you may assume that no one else is accessing hctable at the same time as you (ie, do not worry that someone updated hctable in the time between your reading the contents of hctable and your updating hctable).

(page intentionally blank)

```
const express = require('express');
const { Pool } = require('pg');
const app = express();
const port = 3001;
// PostgreSQL connection configuration
const pool = new Pool({
user: 'XXXXX,
host:'127.0.0.1',
database: 'hc',
password: 'Perrybuy123456$',
port: 5432,
});
// Function to calculate Fibonacci number
function fibonacci(num) {
let a = 0, b = 1, sum = 0;
for (let i = 2; i <= num; i++) {
sum = a + b;
a = b;
b = sum;
}
return num ? b : a;
}
// Route to handle "/page" request
app.get('/page', async (req, res) => {
try {
const client = await pool.connect();
// Query the current count from the database
const result = await client.query('SELECT count FROM hctable');let count =
result.rows[0].count;
// Increment the count and update the database
count++;
await client.query('UPDATE hctable SET count = $1', [count]);
// Calculate the Fibonacci number based on the updated count
const fibNumber = fibonacci(count);
// Return an HTML page with the current count and the Fibonacci number
res.send(`<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Page Count and Fibonacci</title>
</head>
<body>
<p>This page has been requested ${count} times.</p>
<p>The ${count}th Fibonacci number is ${fibNumber}.</p>
</body>
</html>`);
// Release the client back to the pool
client.release();
} catch (error) {
console.error(error);
res.send("Error occurred");
}
});
// Start the server
app.listen(port, () => {
con
```