Name: _____

Please answer the following questions within the space provided on the following pages. Should you need more space, you can use scratch paper, but clearly label on the scratch paper what problem it corresponds to. While you are not required to explain your queries, comments may help me to understand what you were trying to do and thus increase the likelihood of partial credit should something go wrong. If you get entirely stuck somewhere, explain in words as much as possible what you would try.

This is a pen and paper exam, and thus computers and internet capable devices are prohibited. If you have any confusion about question intention or wording, please do not hesitate to ask!

Your work must be your own on this exam, and under no conditions should you discuss the exam or ask questions to anyone but myself. Failure to abide by these rules will be considered a breach of Willamette's Honor Code and will result in penalties as set forth by Willamette's academic honesty policy.

Please sign and date the below lines to indicate that you have read and understand these instructions and agree to abide by them. *Failure to abide by the rules will result in a 0 on the test.* Good luck!!

_____ _____

Signature Date

Question:	1	2	3	4	5	Total
Points:	3	8	6	15	18	50
Score:						

1. Consider a table created with the following query, which has then been populated to have 5 million rows.

```
CREATE TABLE to_index_or_not (
  id SERIAL PRIMARY KEY,
  name TEXT,
  birthday DATE,
  favorite_num INT
);
```

The question you are faced with is whether it would be worthwhile to add an index to the birthday column, via:

```
CREATE INDEX bdindex ON to_index_or_not (birthday);
```

For each of the below queries, run today, indicate whether the index would be useful for speeding up *that particular query.*

(1) (a)
```
SELECT name
FROM to_index_or_not
WHERE birthday = 'Jan 5, 2014'
```

A. Index useful
B. Index not useful

(1) (b)
```
SELECT name, birthday
FROM to_index_or_not
WHERE favorite_num = 8
```

A. Index useful
B. Index not useful

(1) (c)
```
SELECT MIN(name)
FROM to_index_or_not
WHERE birthday < 'October 29, 2024'
```

A. Index useful
B. Index not useful

2. On the last page of this exam are three tables that you will utilize for both problems 2 and 4. I put them on the last page so that you could rip it off to use as an easier reference. Looking at tables `tab1`, `tab2` and `tab3`, for each of the following queries, determine whether that query would run successfully or not. If it does not, explain directly what error would occur and why it occurred.

(2) (a)
```
INSERT INTO tab3 VALUES
(7, -4, 'Katy', '2022-02-14');
```

Solution: This should be fine

(2) (b)
```
UPDATE tab2
SET E = -4.83;
```

Solution: This should be fine, as -4.83 is a valid entry in the referencing tab1.

(2) (c)
```
ALTER TABLE tab3 ALTER COLUMN G SET DATA TYPE INT;
```

Solution: This will cause an error, as the -0.24 will be rounded to 0 at which point it breaks the $F > G$ constraint.

(2) (d)
```
DELETE FROM tab1
WHERE A = 'Henry' AND B = 0;
```

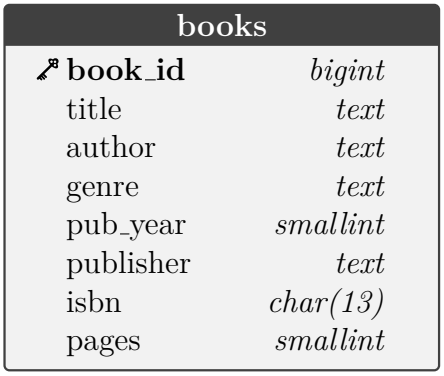
Solution: This will be ok, because nothing is currently referencing the -4.83 that corresponds to the Katy row.

(3) 3. (a) What normal form is the below table currently in? Justify your answer for full points.

🔑 course_id	course_name	course_type	cert_req
201	Safety Protocols	Safety	Yes
202	Advanced Welding	Technical	Yes
203	Customer Relations	Soft Skills	No
204	Machine Maintenance	Technical	Yes
205	Leadership Basics	Soft Skills	No
206	Conflict Resolution	Soft Skills	No
207	Electrical Safety	Safety	Yes
208	Chemical Handling	Safety	Yes
209	Basic Programming	Technical	Yes
210	Communication Strategies	Soft Skills	No

Solution: We seem to have just one piece of information in each cell, and a primary key that correctly and uniquely denotes each row. So we are in 1st normal form. Since we don't have a compound primary key, we are thus also immediately in 2nd normal form, since partial dependencies are impossible. However, we do not seem to be in 3rd normal form, as there looks to be a transitive dependency between `cert_req` and `course_type`.

(3) (b) Below are the ERD summary of a table and the unnormalized data that should be inserted into it. What normal form is this table in? Justify your answer for full points.




```
[
  {
        "BookID": 1,
        "Title": "The Little Prince",
        "Author": "Antoine de Saint-Exupery",
        "Genre": "Children's",
        "PublicationYear": 1943,
        "Publisher": "Reynal & Hitchcock",
        "ISBN": "9780156012195",
        "PageCount": 96
  },
  {
        "BookID": 2,
        "Title": "To Kill a Mockingbird",
        "Author": "Harper Lee",
        "Genre": "Young Adult",
        "PublicationYear": 1960,
        "Publisher": "J.B. Lippincott & Co.",
        "ISBN": "9780061120084",
        "PageCount": 281
  },
  {
        "BookID": 3,
        "Title": "1984",
        "Author": "George Orwell",
        "Genre": "Dystopian Fiction",
        "PublicationYear": 1949,
        "Publisher": "Secker & Warburg",
        "ISBN": "9780451524935",
        "PageCount": 328
  }
]
```


Solution: Again, it looks like we are in 1st normal form, as we have a valid and unique primary key and each cell could have a single piece of information. And just like before, we thus get 2nd normal form for free, as there can be no partial dependencies. So we just have to check for transitive dependencies. Here, it looks like we do not have any. An argument could be made that things like `PageCount` depend on the `ISBN`, but the `ISBN` and the `BookID` seem to be serving similar roles here. So at least with the data we have, it would probably be safe to say it was in 3rd normal form.

4. Here, we will reuse the same tables as from Problem 2 (assuming none of the alterations from Problem 2 were done) located on the last page of this test. Use these tables to compute the output of the following queries. Output should include column names, as well as content.

(5) (a)
```
SELECT tab2.D, AVG(tab3.F) AS avg_f
FROM tab3
RIGHT JOIN tab2
   ON tab2.D = tab3.I
GROUP BY tab2.D
ORDER BY tab2.D;
```

Solution:

d	avg_f
2022-01-30	1
2022-02-14	(NULL)
2022-04-24	8

(5) (b)
```
SELECT COUNT(*) - MAX(tab1.B + tab3.F) AS value
FROM tab1
FULL OUTER JOIN tab3
  ON tab1.B = tab3.F;
```

Solution:

value
2

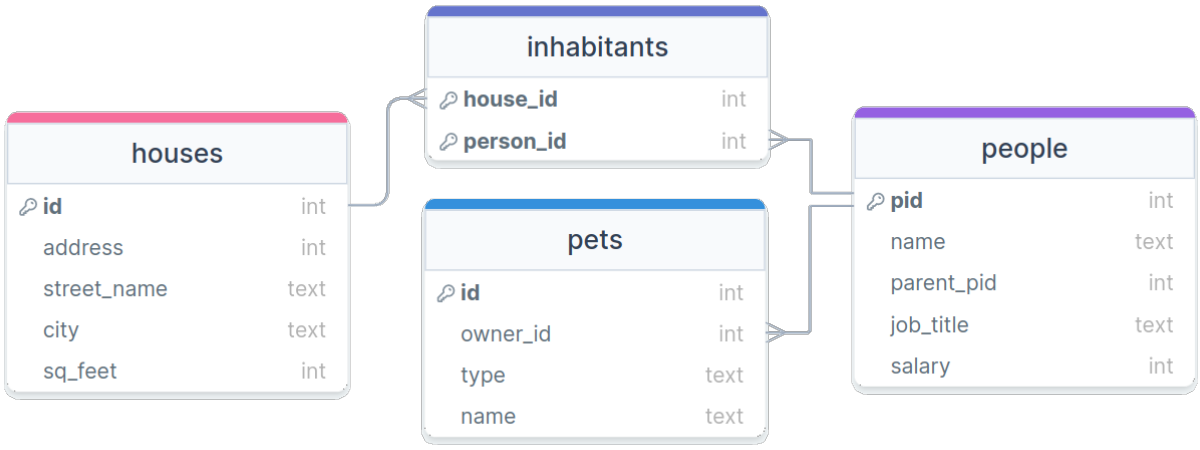
(5) (c)

```
SELECT AVG(tab1.B) + COUNT(tab2.D) AS num
FROM tab1
LEFT JOIN tab2
  ON tab1.C = tab2.E
JOIN tab1 AS ttab1
  ON ttab1.B > tab1.C;
```

Solution: Again, really should have given this a column alias, so I'll take any column name.

num
3.0

5. You have gathered together some census data on people living in a variety of different housing situations, resulting in the collect of the 4 tables shown below. This ERD is also included on the back of the last page, for easier reference while working on these problems.



Given this collection of tables, where primary and foreign keys have been defined as shown, write out a query that would answer each of the following questions. All of these questions *can* be answered with a single query (and no subqueries), but feel free to use multiple queries if you want. Just make sure it is clear to me what you are doing.

(6) (a) How many houses have nobody living within them?

Solution: Here we are looking for the absence of data, so we'll use the left/right join and then filtering by NULLs trick.

```
SELECT COUNT(*)
FROM houses
LEFT JOIN inhabitants
  ON houses.id = inhabitants.house_id
WHERE inhabitants.house_id IS NULL
```

(6) (b) What are the names of the `'CEO'`'s who own more than 3 pets?

Solution: This is just knowing how to use GROUP BY and HAVING

```
SELECT people.name
FROM people
JOIN pets
  ON people.pid = pets.owner_id
WHERE people.job_title = 'CEO'
GROUP BY people.pid, people.name --names unique w/in pid
HAVING COUNT(*) > 3
```

(6) (c) How many different households have a parent living with a child?

Solution: Here we basically look at parent children pairs and then work out what house each inhabits to see if they are the same.

```
SELECT COUNT(*)
FROM houses AS h_child
JOIN inhabitants AS i_child
  ON h_child.id = i_child.house_id
JOIN people AS child
  ON i_child.person_id = child.pid
JOIN people AS parent
  ON child.parent_pid = parent.pid
JOIN inhabitants AS i_par
  ON parent.pid = i_par.person_id
JOIN houses AS h_par
  ON h_par.id = i_par.house_id
WHERE h_par.id = h_child.id
```

Tables for problems 2 and 4:

tab1

```
CREATE TABLE tab1 (
  A TEXT,
  B INT CHECK (B >= 0),
  C NUMERIC(4,2) UNIQUE,
  PRIMARY KEY (A, B)
);
```

A	B	C
Henry	2	4.23
Jake	1	10.52
Henry	0	5.17
Katy	3	-4.83
Jake	4	83.10

tab2

```
CREATE TABLE tab2 (
  D DATE PRIMARY KEY,
  E NUMERIC(4,2)
    REFERENCES tab1 (C)
);
```

D	E
2022-02-14	10.52
2022-01-30	83.10
2022-04-24	4.23

tab3

```
CREATE TABLE tab3 (
  F INT PRIMARY KEY,
  G REAL,
  H TEXT,
  I DATE
    REFERENCES tab2 (D)
  CHECK (F > G)
);
```

F	G	H	I
2	1.1	Jake	2022-01-30
8	6.445	Katy	2022-04-24
0	-0.24	Henry	2022-01-30

ERD from Problem 5, repeated for ease of access: