CS 348, Spring 2022 - Homework 5: Functional Dependencies, Normalization, and Indexes.

(100 Points)

Due on: **4/1/2022 at 11:59 pm**
This assignment is to be completed by individuals. You should only talk to the instructor, and the TA about this assignment. You may also post questions (and not answers) to Campuswire.

There will be a 10% penalty if the homework is submitted 24 hours after the due date, a 20% penalty if the homework is submitted 48 hours after the due date, or a 30% penalty if the homework is submitted 72 hours after the due date. The homework will not be accepted after 72 hours, as a solution will be posted by then.

Submission Instructions: Write your answers for Questions 1, 3, and 4 in a word/text file and generate a pdf file. **Upload the pdf file to Gradescope**. For Question 2, write your queries **in Q2.py and upload the file to Brightspace**.

Question 1) Functional dependencies and normalization.

a) Create an example relation that has two troublesome FDs. The first troublesome FD should be a partial dependency FD that violates 2NF. The second troublesome FD is a transitive FD that violates 3NF, but not 2NF. Your example should be different from the FD and normalization examples we had in the lectures, homework, and quizzes. You can select any domain for your relation (e.g., medical data, social networks, geography, history, e-commerce, … etc.). A source of inspiration I usually use is Wikipedia list-of pages (e.g., list of languages or list of candy bars). (5 points)

   https://en.wikipedia.org/wiki/List_of_lists_of_lists

Ans: <Database: Student> - troublesome **FD violation of 2NF**

Student(student_id, department_id, student_name, department_name, student_info, tuition)

According to the definition of 2NF, all non-key attributes must depend on a whole key. From my example of student database,

Student_id, department_id -> student_info

Student_id -> student_name

Department_id -> department_name

Student_info -> tuition

- **student_name** is functionally dependent on a subset of a key **student_id.**
- **department_name** is functionally dependent on a subset of a key **department_id.**

Therefore, the <**Student**> relation violates the 2NF in Normalization and is considered a bad database design.

- Ans: <Database: Student> - Troublesome FD **violation of 3NF, but not 2NF**

According to the definition of 3NF, table is in 2NF and all non-key attributes must depend on only a key. From my example of student database,

Student_info -> tuition

'Student_info' is a transitive functional dependency in the relation, and the relation is violating 3NF because grade is not a key.

b) In which normal form is your table in? (2 points)

- Ans: Because the table, Student, is violating 2NF, but passes Student_id -> student_name, it is **1NF**.

c) Show a small instance of your table. (2 points)

Ans:

Student_id	Department_id	Student_name	Department_name	Student_info	tuition
101	1	Junseok Oh	Science	Sci_sophomore	30000
102	2	Suzy Bae	Communication	Com_senior	25000
103	1	Minwoo Jung	Science	Sci_senior	25000

d) Write one legal Update or Insert statement that violates one of the FDs. List two rows of your table that show the violated FD. (4 points)

- Ans: INSERT INTO Student Values (104, 1, "Alex Choi", "Communication", "com_sophomore", 27500);

Student_id	Department_id	Student_name	Department_name	Student_info	tuition
101	1	Junseok Oh	Science	Sci_sophomore	30000
104	1	Alex Choi	Communication	com_sophomore	27500

- Ans: Rows with 101 and 104 department_ids are showing violated FD because department_name is dependent upon department_id. However, two them have same department_name but different department_id.

e) Decompose your relation in point (a) to BCNF. (5 points)

- Ans:

Student (Student_id, Department_id, Student_name, Student_info)

Department (Department_id, Department_name)

Tuition (Student_info, Tuition)

f) Is your decomposition lossy or lossless? (2 points)

- Ans: It is a lossless.

g) Show an example of a lossy decomposition of your relation in point (a). Explain briefly why your decomposition is lossy. (5 points)

Student (Student_id, Department_id, Student_name)

Department (Department_id, Department_name)

Tuition (Student_info, Student_name, Tuition)

- Ans: When decomposing from Student table into the Tuition table, a key Student_name, is not a key for either of these tables. Therefore, its decomposition is lossy.

Question 2) Data cleaning using automatic functional-dependency recognition:

Suppose you are responsible for the data lake[1] of your company. The data is not clean (e.g., incorrect or missing values). You need to analyze the data with the help of the concept of functional dependencies to aid your team in the cleaning process. For example, consider the following table with two attributes A and B:

A, B
1, a
2, b
2, b
2, b
2, c
2, q
3, d
3, d
6, c
6, c
6, c
6, b

We notice that the FD A -> B almost holds. There are only a few rows that violate this FD (e.g., (6,b)). It is possible that the value 'b' is just an incorrect data item and should be corrected to 'c'. Finding similar instances in a large number of files with large number of rows and columns is time consuming. We can use SQL to find those cases and report them to a data analyst for further investigation.

[1] https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/

a) Finding the correct value for B for each A value. (10 points)
   Write a query to find the correct value for B for each A value. For A = X, the correct B value is the one with the largest number of rows where A = X (e.g., for A=2, the value 'b' is the correct value since it has the largest number of rows in the A=2 group). For the provided table, your query should return the following result. You can assume that for one A value, there is only one value in B that has

a majority.

A	B
1	a
2	b
3	d
6	c

b) Providing statistics for FD violations (15 points).
Write an SQL query to provide the following statistics for an FD. For each value in the A column, return the number of rows that satisfy the FD and the number of rows that violate the FD. For example, consider the value A=2 in the table above. The value B= 'b' is the most popular for A=2. Therefore, we conclude that for A=2 there are three correct rows and two incorrect rows ((2, 'c') and (2, 'q')).

Expected result based on the given example table:

A	B	no_correct_rows	no_incorrect_rows
1	a	1	0
2	b	3	2
3	d	2	0
6	c	5	1

Note: The zeros in the above result were Null values converted using the ifnull function in SQLite.

Instructions for Question 2: You can use the SQLite database question2.db to test your queries. Include your answers (queries) in Q2.py and submit the file to Brightspace.

Question 3 (25 points, about 3 each):

Assume we have a table for homes/houses information:

```
homes(home_ID, owner_ID, type, price, no_floors, sq_feet)
Where home_ID is a key.
```

For data distribution we will assume the following values (based on Zillow prices for the Lafayette and W. Lafayette area).

Type is either ‘for rent’ or ‘for sale’

no_floors (number of floors) is 1, 2, or 3

price range is 500 to 3500 for renting (type = ‘for rent’) and 60,000 to 1,300,000 for buying (type =’for sale’).

sq_feet is from 1000 to 9000

Consider the following queries:

1. ```
   Select * from homes
   Where home_ID = 12345;
   ```
2. ```
   Select * from homes
   Where owner_ID = 12345;
   ```
3. ```
   Select * from homes
   Where sq_feet >= 1000 AND sq_feet <= 8000;
   ```
4. ```
   Select * from homes
   Where price >= 200,000 AND price <= 205,000;
   ```
5. ```
   Select * from homes
   Where no_floors >= 1 AND no_floors <= 2;
   ```
6. ```
   Select * from homes
   Where price >= 200,000 AND price <= 210,000
     AND type = 'for sale';
   ```
7. ```
   Select * from homes
   Where owner_ID = 12345 OR price =3000;
   ```
8. ```
   Select * from homes
   Where type = 'for rent';
   ```

a) Suppose that you are allowed to create only **one hash index, one clustered B+tree index, and one unclustered B+tree index** on the homes table. Each index (hash or B+tree) can be on a single attribute or two attributes. Pick indexes that help as many queries as possible. Your indexes should support the queries that need indexes the most.

- Ans:
- One hash index: <home_id>
- One clustered B+tree index : <price, sq_feet>
- One unclustered B+tree: <owner_id>

b) For each query, list the following:

   I. Index(s) that can support the query. If multiple indexes can be used then briefly describe how the indexes can be used together.

   II. How useful each index you include in the previous point (very useful, somewhat useful). Very useful means that for the query the data entries will be adjacent in the index.

   III. If the query should not be supported with any indexes, then describe why you think so.

1.
```
Select * from homes
Where home_ID = 12345;
```

Ans:

I. Hash index <Home_ID>

II. Very useful

III. This query requires index.

2.
```
Select * from homes
Where owner_ID = 12345;
```

Ans:

I. Unclusterd B+ Tree <Owner_ID>

II. Very useful

III. This query requires index.

3.
```
Select * from homes
Where sq_feet >= 1000 AND sq_feet <= 8000;
```

Ans:

I. Clustered B+ Tree <price, sq_feet>

II. somewhat useful

III. This query requires index.

4.
```
Select * from homes
Where price >= 200,000 AND price <= 205,000;
```

Ans:

I. Clustered B+ Tree <Price, sq_feet>

II. very useful

III. This query requires index.

5.
```
Select * from homes
Where no_floors >= 1 AND no_floors <= 2;
```

Ans:

I. This query should not use supported indexes because there are only three different values for no_floors (1, 2, 3), so it is unnecessary to make an index to support this query.

6.
```
Select * from homes
Where price >= 200,000 AND price <= 210,000
  AND type = 'for sale';
```

Ans:

I. Clustered B+ Tree <price, sq_feet>

II. somewhat useful

III. This query requires index.

7.
```
Select * from homes
Where owner_ID = 12345 OR price =3000;
```

Ans:

I. Clustered B+ Tree <owner_id, price>, uncluster b+ tree on <owner_ID>

   a. We have to have two separate data with owner_id leaf nodes, and distribute with price leaf.

II. somewhat useful

III. This query requires index.

8.
```
Select * from homes
Where type = 'for rent';
```
Ans:

I. This query should not use supported indexes because there are only two different values for type (for rent or sale), so it is unnecessary to make an index to support this query.

Question 4:

It is sometimes possible to evaluate a particular query using only indexes, without accessing the actual data records. This method reduces the number of Page IOs and hence speed up the query execution.

Consider a database with two tables:

Product(p_ID, name, category, rating, price)

Assume two unclustered indexes, where the leaf entries have the form [search-key value, RID] (i.e., alternative 2).

I1: <rating> on Product

I2: <price> on Product

For the following queries, say which queries can be evaluated ***with just data from these indexes***.

- If the query can, describe how by including a simple algorithm.
- If the query can't, briefly explain why.

a. (zero points, answer is included)
SELECT MIN(rating)
FROM Product;
Solution: The query can be evaluated from the <rating> index. The query result is the search-key value (rating) of the leftmost data entry in the leftmost leaf page.

b. (5 points)
SELECT COUNT(distinct rating)

FROM Product;

ANS: The query can be evaluated from the <rating> index. The query result is the search-key value (rating) of the leftmost data entry in the leftmost leaf page.

c. (5 points)

SELECT AVERAGE(price)
FROM Product
GROUP BY category;

ANS: The query cannot be evaluated from the <price> index with grouping with category. The result, 'price,' from this query is not useful because we cannot know which category has which average price without accessing the actual data records.

d. (7 points)

SELECT AVERAGE(price)
FROM Product
GROUP BY p_ID;

ANS: The query can be evaluated from the <price> index. The query result is the search-key value (price) of the leftmost data entry in the leftmost leaf page.

The group with p_ID is an unique key value, so its order of the table won't get affected so we don't have to actually access to the data.

e. (8 points)

SELECT rating, AVERAGE(price)
FROM Product
GROUP BY rating;

ANS: The query can be evaluated from the <rating, price> index. The query result is the search-key value (rating, price) of the leftmost data entry in the leftmost leaf page. With selecting an index 'rating', which is also grouped by, and average of price will allow evaluate without accessing an actual data.