Connor Brown

CS 348 - Homework 4: Functional Dependencies, Normalization, and Indexes.

(100 Points)

Fall 2020
Due on: **11/06/2020 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 15% penalty if the homework is submitted 48 hours after the due date, or a 20% 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 Instruction: Write your answers on this word file and generate a pdf file. **Upload the pdf file to Gradescope**.

Question 1:
Consider the following relation about blood-sugar readings:

BSR(PatientID, Date, PatientName, PatientAge, SugarLevel, InsulinDose)
The BSR relation has the following functional dependencies:
PatientID, Date -> SugarLevel
PatientID -> PatientName, PatientAge
SugarLevel -> InsulinDose.

To get an idea of the domain of sugarLevel and insulinDose attributes, you can look at the following table:
https://www.researchgate.net/figure/The-recommended-insulin-dosage-according-to-the-patients-glucose-level-Blood-sugar_tbl1_262295520

a. Give an example instance where the function dependency SugarLevel -> InsulinDose is violated (include only two or three rows). (5 points)
    a. **Given a case where Sugar Level is 140 it is possible to get two different insulin doses. Say it is possible to get the dose of 0 or 2. This would**

cause the function dependency SugarLevel->InsulinFose to be violated.

b. In what normal form is the above relation? (5 points)
    a. **The relation above is in 1NF.**
c. Decompose the BSR relation into BCNF. **In each step of your decomposition, show the FDs you are lifting and the resulting tables.** (5 points)
    a. **2NF:**
        i. **Lift PatientID -> PatientName, PatientAge**
        ii. **New-BSR(<u>PatientID, Date</u>, SugarLevel, InsulinDose)**
        iii. **PatientInfo(<u>PatientID</u>, PatientName, PatientAge)**
        iv. **Functional Dependencies:**
            1. **PatientID, Date -> SugarLevel**
            2. **SugarLevel -> InsulinDose.**
    b. **BCNF:**
        i. **Lift SugarLevel -> InsulinDose**
        ii. **New-BSR(<u>PatientID, Date</u>, SugarLevel)**
        iii. **PatientInfo(<u>PatientID</u>, PatientName, PatientAge)**
        iv. **DoseLevels(<u>SugarLevel</u>, InsulinDose)**
d. Show an example of a lossy decomposition of BSR. Explain briefly why your decomposition is lossy. (5 points)
    a. **Lossy Decomposition**
        i. **New-BSR(<u>PatientID, Date</u>, InsulinDose)**
        ii. **PatientInfo(<u>PatientID,</u> PatientName, PatientAge)**
        iii. **DoseLevels(<u>SugarLevel</u>, InsulinDose)**
        iv. **Functional Dependency:**
            1. **PatientID, Date -> SugarLevel**
    b. **Explanation**
        i. **This is a lossy decomposition as by doing a natural join between the three tables you will end up with extra tuples. This decomposition causes there to be several extra tuples for each bsr because it will list copies where InsulinDose is the same for different SugarLevel.**

Question 2:

Suppose the BSR relation in Question 1 has hundreds of thousands of rows – too many for a person to read. We want to know whether or not this table currently satisfies the functional dependency SugarLevel -> InsulinDose. This is critical data

where checking such FD can be lifesaving. Explain how to use one or more SQL queries to test if this FD is satisfied in the relation. Make sure you explain how the results of your queries determine whether the FD is satisfied (e.g., if two queries return different number of rows then the FD is violated).

a. Give a solution using the aggregate function COUNT. (10 points)
    a. **SELECT ***

       FROM BSR

       GROUP BY SugarLevel

       HAVING COUNT (DISTINCT InsulinDose) > 1;

    b. **If a SugarLevel is listed, then it violates the FD. This is because if it lists a SugarLevel and it has more than one InsulinDoses associated with it then it fails.**
b. Give a solution using GROUP BY and HAVING. (10 points)
    a. **SELECT ***

       FROM BSR

       GROUP BY SugarLevel

       HAVING COUNT (DISTINCT InsulinDose) > 1;

    b. **If a SugarLevel is listed, then it violates the FD. This is because if it lists a SugarLevel and it has more than one InsulinDoses associated with it then it fails.**
c. Give a solution that joins the BSR relation to itself (no GROUP BY, HAVING, and aggregate functions). (10 points)
    a. **SELECT A.SugarLevel, B.SugarLevel**

       FROM BSR A, BSR B

       WHERE WHERE A.SugarLevel <> B.SugarLevel AND A.InsulinDose = B.InsulinDose

       ORDER BY A.InsulinDose

    b. **If an InsulinDose has the same SugarLevel listed in 2 different doses, then it violates the FD.**

Question 3:

Assume we have a table that includes student information:

```
Student(ID, username, name, Age, Height)
    Where ID is a key, and username is a key
```

There are four indexes on this table:

I1: Unclustered hash index on <ID>

I2: Unclustered B+ tree index on <username>

I3: Clustered B+ tree index on <name>

I4: Unclustered B+ tree index on <Age>

I5: Unclustered B+ tree index on <Height>

For each of the WHERE conditions below, say which index you think is best to use and **why** you think it's the best. **Choose only one index.**

a. (5 points):

```
Select * From Student
Where username="theLuckyOne" AND name Like 'a';
```

Index 2 would be best for this as we are looking for an exact key value from the username. And with the username as a key we can easily find it.

b. (8 points):

```
Select * From Student
Where (name >= 'a' AND name < 'b')
       AND (username >= 'a' AND username < 'b');
```

Assume that the number of students having a name that starts with the letter 'a' is the same as the number of students with a username that starts with the letter 'a'.

Index 3 would be best as this search index is asking us to select students whose name starts with an 'a' and username starts with an 'a'. It is the best because it is clustered and is wanting

several values that would all be stored next to each other. So, it would only need to traverse the left side of the tree.

c. (10 points):

```
Select * From Student
Where (Age < 18 OR Age = 38 OR Age > 50)
       AND (Height > 6.5)
```

Assume the number of students having an age in the specified range is the same as the number of students whose height is larger than 6.5.

Index 5 would be the best index to use. It allows us to do a range query based on the age of students still but makes it so that we traverse a smaller portion of the tree. If we were to use index 4, we would have to traverse more parts of the tree in order to see check the range of ages. Index 5 allows us to stay on the right side of the binary tree, but index 4 would have us traverse the left, middle, and right side. So Index 5 is the best index to use.

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:

Book(ISBN, title, year, publisher)

Sells(ISBN, vendor, price)

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

I1: <year> on Book
I2: <publisher> on Book
I3: <price, ISBN> on Sells

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(price)
   FROM Sells;
   Solution: The query can be evaluated from the <price, ISBN> index. The query result is the price part of the search-key value of the leftmost leaf entry in the leftmost leaf page.

b. (5 points)
   SELECT AVERAGE(price)
   FROM Sells;

The query can be evaluated from the <price, ISBN> index. The query is the result from traversing the B+ tree and adding the price part of the search-key values together, then dividing by the number of data records.

c. (4 points)
   SELECT AVERAGE(price)
   FROM Sells
   GROUP BY vendor;

The query cannot be evaluated form any of the indexes provided. This is because the only index that involves the price is index 3. From index 3 we can get the ISBN and price. But based on the ISBN we cannot get the vendor. In a case where we have the same ISBN and price at separate vendors it would not be able to give us the correct data from the index.

d. (8 points)
   SELECT COUNT(*)
   FROM Book
   WHERE publisher = 'Knopf' AND year = 2010;

The query can be evaluated from the <publisher> index and <year> index. After using the <publisher> index and <year> index you can retrieve the records where the RID is in both sets. Then count the total number of entries.

e. (10 points) (think carefully about this one!):
   SELECT ISBN, AVERAGE(price)

```
FROM Sells
GROUP BY ISBN
HAVING COUNT(DISTINCT vendor) > 1;
```

This can be evaluated from the <price, ISBN> index. We first sort the RIDs by ISBN. We then filter out the ISBN who do not have more than 1 distinct vendor. After we take the average price for each group of ISBN values.