CS348 HW5 Spring-2024

CS 348, Spring 2024 - Homework 5: Functional Dependencies, Normalization, and Indexes.

(100 Points)

Due on: **April 3, 2024 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 Ed.

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, 2.a-2.b, 3, and 4 in a word/text file and generate a pdf file. **Upload the pdf file to Gradescope**. Your word/pdf file should follow the following format (i.e., do not include any question text):

Q1.a

Q1.b

Q1.c

…

Q2.a

Q2.b

…

For Questions **2.c, 2.d, and 2.e, you will be able to upload your answers to Gradescope** (similar to Homework 1 and 2). Further instructions will be posted in Ed.

Q1) Functional dependencies and normalization.

a. (10 points) Create one relation R1 with a troublesome FD (a partial dependency FD) that violates 2NF. Create another relation R2 with a troublesome FD (a transitive FD) that violates 3NF, but not 2NF. Your relations must be in the domain of the Homework1 database (e.g., bike stations, bikes, trips, .. etc.). You can include attributes that are not in the Homework1 database.

b. (2 points) Show a small instance of R1.

c. (3 points) Write one legal Update or Insert statement that violates the FD in R1. List two rows of your table that show the violated FD.

d. (6 points) Decompose R1 and R2 to BCNF.

e. (1 point) Is your decomposition lossy or lossless?

f. (3 points) Show an example of a lossy decomposition of R1.

Q2) (25 pts) Functional dependencies, data cleaning, and temporal databases.

A copy of the SQLite database used for this question is included with this homework (employee.db).

Table instance example:

id	**name**	**salary**	**dept_id**	**dept_name**	**timestamp**
1	John	55000	101	Sales	100
1	John	56000	101	Sales	101
....					
4	Ali	60000	101	Sales	100

4   Ali   61000   101   Sales   105

Temporal databases* keep track of old versions of a table. For example, when an update happens the old row is kept and a new row, that reflects the new update, is inserted. In this simple design, each row has different versions at different timestamps, such as in the table above. The larger the timestamp the more recent is the row version.

* https://en.wikipedia.org/wiki/Temporal_database

Note: You can use the SQLite database employee.db to test your queries. Grading will use a different instance. Instructions for submitting Question 2.c-q2.e will be posted in Ed later this week.

a. (2 points) In the table shown above, the FD [(id, dept_id) -> dept_name] is troublesome and can be violated. Describe a graphical user interface for data entry that allows an end user to easily violate the FD.

b. (1 point) With the timestamp attribute added to the table, the id is no longer a key. Pick a new primary key for the table.

c. (6 points) One interesting feature of temporal databases is to query the database and show a table snapshot at a specific time t. For a time t and a row r, a table snapshot contains the version of r at time tr where tr is the largest timestamp that satisfies the condition tr <= t. A row w inserted after t must not be included in the result. A table snapshot at time t can be helpful in some scenarios, such as in auditing and investigations (e.g., if someone updated the table to hide/erase some data on purpose!).

For example, at time 106 the employee table had the following rows:

```
id name salary dept_id  dept_name timestamp
--  -------- -------- ----------- -----------------  -------------
1  John   57000    103          HR           106
3  Van     62000    101        Sales         101
4   Ali     61000    101        Sales         105
2  Sara   57000     102      Marketing      105
```

Write a query to list the rows in the table at time 103.

d. (8 points) Find the first time the FD [(id, dept_id) -> dept_name] was violated (the smallest/earliest timestamp t2 where (id, dept_id) was similar to a previous timestamp t1, but the dept_name changed unexpectedly).

Expected Result:

```
id  t1  t2   dept_id dept_name_at_t1 dept_name_at_t2
--  ---   ---  ---------- ------------------------ -------------------------
1 100 107   101         Sales                 Marketing
2 105 120   102      Marketing                Finance
```

e. (8 points) Fix the rows violating the FD [(id, dept_id) -> dept_name]. For each (id, dept_id) use the oldest dept_name value. Write a query to return the correct data.

Expected Result:

```
id name salary dept_id  dept_name timestamp
--  -------- -------- ----------- -----------------  -------------
1  John    55000   101        Sales          100
1  John    56000   101        Sales          101
1  John    56000   102      Marketing        102
1  John    56000   103         HR            106
1  John    56000   101        Sales          107
1  John    56000   103         HR            115
2  Sara    56000   101        Sales          100
2  Sara    56000   104       Finance         103
2  Sara    56000   105         IT           104
2  Sara    56000   102      Marketing        105
2  Sara    56000   102      Marketing        120
3  Van     56000   101        Sales          101
4   Ali    60000   101        Sales          100
4   Ali    61000   101        Sales          105
```

Q3) Consider the inventory database of an electronic store. The devices table in the database has the following schema:

devices (item_id, item_type, item_brand, item_price, item_quantity_available, items_sold, item_last_ordered)

You can consider item_id to be the primary key for this table. You need to pick three indexes (one clustered B+ Tree index, one unclustered B+ Tree index and one hash index) to create on this table to support the maximum number of queries. The queries are supplied below (a-f). Here are some additional statistics about the table that might aid you in your choices.

Useful Statistics:

- There are 10000 devices in the inventory, each device having a unique id that ranges between (1 - 10000).

- The item_type is amongst the following (‘Mobile phone’, ‘Laptop’, ‘Tablet’, ‘TV’, ‘Monitor’, ‘Cables’, ‘PC’)

- The electronic store specializes in selling mobile phones, with most of them priced between 200 and 1000 USD. However, the store also sells bigger electronics like laptops whose prices range between 1000 and 2500 USD. Consider a distribution where phones and laptops account for 75% and 20% of the inventory, respectively, with the remaining 5% comprising various miscellaneous items.

- The devices in the electronic store belong to the following brands (Apple, HP, Samsung, Dell, Lenovo). You can imagine that the brands are evenly distributed across all devices, i.e., ~2000 devices belonging to each brand.

- item_last_order is a date column which denotes the last date on which the item was last ordered by the electronic store to restock its inventory.
- item_quantity_available and items_sold are two numeric columns which denote the available stock in inventory and number of items sold by the store respectively. The items_sold column represents the total sales of each device since it first became available, leading to potentially high values in this column.

For the answer, please list your chosen indexes. Your chosen indexes should support the queries that need indexes the most. Also, for each query, determine whether indexing is suitable for the query or not. If no index is suitable from the list, describe why. If suitable, list the index (from the three indexes you picked) that you think is useful for the query.

Queries (4 points each):

a. Find devices where the item_price ranges between 1000 to 2500 inclusive.

   SELECT * FROM devices WHERE item_price >= 1000 and item_price <= 2500;

b. List the device with item_id = 5603.

   SELECT * FROM devices WHERE item_id = 5603;

c. Select all the devices whose item_type = ‘PC’

   SELECT * FROM devices WHERE item_type = ‘PC’;

d. Select all the Apple, Samsung and HP devices:

   SELECT * FROM devices WHERE item_brand in (‘Apple’, ‘Samsung’, ‘HP’)

e. Find the tablet devices whose price < 300:

   SELECT * FROM device WHERE item_price < 300 AND item_type = ‘Tablet;

f. Obtain all the devices in which over 50 units have been sold:

   SELECT * FROM devices WHERE items_sold > 50;

Q4) (26 pts)

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 speeds up the query execution time.

Consider a database with two tables:

Console (**<u>cid</u>**, cname, manufacturer, price, color)

Game (**<u>cid, gid</u>**, year)

Game.cid is a foreign key to Console.cid

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

I1: <price> on Console relation

I2: < color > on Console relation

I3: <year, cid> on Game relation

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

- If the query can be evaluated with the data from the indexes,
  - Firstly, clearly mention *Yes.*
  - Secondly, describe how by including a simple algorithm.
- Else if the query can't be evaluated with the data from the indexes,
  - Firstly, clearly mention *No.*
  - Secondly, briefly explain why.

a. (zero points, answer is included)
   SELECT MIN (price)
   FROM Console

☐ Answer: Yes.

☐ Explanation: The query can be evaluated from the I1 index on <price>. The query result is the search-key value of the leftmost leaf entry in the leftmost leaf page.

b. (5 pts)
   SELECT DISTINCT (cid)
   FROM Game
   WHERE year = 2024

c. (5 pts)
   SELECT manufacturer, count(*)
   FROM Console
   GROUP BY manufacturer;

d. (5 pts)
   SELECT COUNT(*)
   FROM Console
   WHERE price >= 100 OR color = 'red';

e. (5 pts)

```
SELECT price, COUNT (distinct color)
FROM Console
GROUP BY price;
```

f. (6 pts)

```
SELECT cid, MIN (year)
FROM Game
GROUP BY cid
HAVING COUNT (DISTINCT gid) = 1;
```