CS 122A: Introduction to Data Management – Spring 2016, UC Irvine

Prof. Chen Li

Homework 7: Indexing (100 points)

Due Date: **Thursday, June 2, 2016 11:45 PM, on EEE**

Submission

For this assignment, you need to create **a PDF file.** The filename should be hw7-groupID.pdf (e.g., hw7-01.pdf). You need to submit this file to the EEE dropbox. Points may be deducted if you don't follow the instructions. All homework assignments should have the student IDs and names of your team members. Remember that all homework assignments should be done in a group. This homework assignment should be submitted on EEE before 11:45 pm on the due date. Only one student in a group should submit the file. Everybody on the team is required to have the finally submitted version. Refer to the following table for the submission guidelines. After the 24-hour grace period, no more submission is allowed on EEE. That is, we will **not** accept assignments after that time. We will publish the solutions at that time for the next assignment. Please get all your work in on time!

Date / Time	Place	Remark
Thursday, June 2, 2016 11:45 PM	EEE dropbox	Due date
Friday, June 3, 2016 11:45 PM	EEE dropbox	24-hour grace period - **10 points will be deducted**

Indexing [100 pts]

As the business has grown, important queries needed for the UCA website have started taking way too long to execute. For example, looking up a customer's reservation record by customer email using a prefix takes too much time now.

In this assignment, you are to use MySQL to query and modify the UCA database and its data and show the results. You will need to load the current data into the UCA database to do this. We assume that you have successfully installed a MySQL instance.

Step 1: To make sure you have the right tables with the right content, re-create and re-populate the database using the provided script (the same script used in HW5) with the following schema:

1. **Airplane** = {registration_number:VARCHAR(10), model_number:VARCHAR(10), purchased_year:INTEGER, manufactured_year:INTEGER,capacity:INTEGER}
2. **Airport** = {IATA_code:CHAR(3), name:VARCHAR(40), airport_city:VARCHAR(20), airport_state:VARCHAR(20)}
3. **Customer** = {cid:INTEGER, ssn:CHAR(9) , gender:VARCHAR(6), email:VARCHAR(30), address_street:VARCHAR(50), address_city:VARCHAR(20), address_state:VARCHAR(20), address_zipcode:CHAR(5)}
4. **Credit_Card** = {cid:INTEGER, card_number:VARCHAR(20), expr_date:CHAR(6)}
5. **Flight** = {flight_number:VARCHAR(8), projected_departure_datetime:datetime, projected_arrival_datetime:datetime, aiplane_registration_number:VARCHAR(10), departure_airport_IATA_code:CHAR(3), actual_departure_datetime:datetime, arrival_airport_IATA_code:CHAR(3), actual_arrival_datetime:datetime}
6. **FlightAttendant** = {faid:INTEGER, phone_number:VARCHAR(20), birthdate:date, ssn:CHAR(9), job_title:VARCHAR(50), address_street:VARCHAR(50), address_city:VARCHAR(20), address_state:VARCHAR(20), address_zipcode:CHAR(5), service_year:INTEGER}
7. **Lounge** = {lid:INTEGER, location:VARCHAR(50), airport_IATA_code:CHAR(3)}
8. **MaintenanceEngineer** = {meid:INTEGER, phone_number:VARCHAR(20), birthdate:date, ssn:CHAR(9), job_title:VARCHAR(50), address_street:VARCHAR(50), address_city:VARCHAR(20), address_state:VARCHAR(20), address_zipcode:CHAR(5), skill:VARCHAR(50)}
9. **OperationStaff** = {osid:INTEGER, phone_number:VARCHAR(20), birthdate:date, ssn:CHAR(9), job_title:VARCHAR(50), address_street:VARCHAR(50), address_city:VARCHAR(20), address_state:VARCHAR(20), address_zipcode:CHAR(5), department:VARCHAR(50)}
10. **Pilot** = {pid:INTEGER, phone_number:VARCHAR(20), birthdate:date, ssn:CHAR(9), job_title:VARCHAR(50), address_street:VARCHAR(50), address_city:VARCHAR(20), address_state:VARCHAR(20), address_zipcode:CHAR(5), since:VARCHAR(50)}
11. **Customer_Reserves_Flight** = {cid:INTEGER, flight_number:VARCHAR(8), projected_departure_datetime:DATETIME, purchased_datetime:DATETIME, purchased_price:DECIMAL(7,2), quantity:INTEGER}
12. **Dish** = {lid:INTEGER, name:VARCHAR(40), price:DECIMAL(6,2)}
13. **DishOrder** = {oid:INTEGER, cid:INTEGER, lid:INTEGER, order_datetime:VARCHAR(10), total_amount:INTEGER}
14. **DishOrder_Contains_Dish** = {oid:INTEGER, lid:INTEGER, name:VARCHAR(40), quantity:INTEGER}

15. **Customer_Reserves_Flight** = {cid:INTEGER, flight_number:VARCHAR(8), projected_departure_datetime:datetime, purchased_datetime:datetime, purchased_price:decimal(7,2), quantity:INTGER}
16. **FlightAttendant_Participates_Flight** = {faid:INTEGER, flight_number:VARCHAR(8), projected_departure_datetime:datetime}
17. **MaintenanceEngineer_Maintains_Airplane** = {meid:INTEGER, Aiplane_registration_number:VARCHAR(10)}
18. **Pilot_Operates_Flight** = {pid:INTEGER, flight_number:VARCHAR(8), projected_departure_datetime:datetime}

Step 2: Write the following queries using SQL and run them on your MySQL instance to collect results.

After you open the UCA website, gradually you notice that the site is starting to run slowly. So it is time to create indexes to accelerate important queries. You want to focus on customer email lookups to find his or her reservation information quickly. You also want to add an index on the Customer_Reserves_Flight table so that queries related to projected_departure_datetime can run faster.

1. [10 pts] Create an SQL query that finds customer id, customer email, flight_number, projected_departure_datetime, and the quantity of tickets for each reservation made by customers with an email starting with letter '**i**'.

2. [10 pts] Create the same query to return the same attributes. The only difference is that the filtering condition is "email ending with **d.com**" instead of "starting with letter **i**".

3. [10 pts] Create the same query as question 1 to return the same attributes. The only change that you need to make is the filtering condition is the projected_departure_datetime is on or after '2015-10-01 00:00:00'.

4. [15 pts] We want to create indexes to speed up three queries above. To make the right decision on the indexes, first check the visual query plan for each query above by copying and pasting your SQLs and clicking the lightning + magnifier button in MySQLWorkBench like the following example. **You are not required to submit the visual query plan**.

Now, submit the textual query plan for each of the three queries. To create a textual query plan, you can use the command "EXPLAIN" in front of each query to see the query plan in text (e.g., "EXPLAIN SELECT email from Customer ..."). After adding "EXPLAIN" to a query, click "Query" -> "Execute (All or Selection) to Text" to generate the output and paste the output as an answer.

a. Query 1)
b. Query 2)
c. Query 3)

5. [10 pts] Write and execute a "CREATE INDEX" statement that creates an index on the Customer.email attribute. The index name should be "ix_Customer_email".

6. [10 pts] Write and execute a "CREATE INDEX" statement that creates an index on the Customers_Reserves_Flight.projected_departure_datetime attribute. The index name should be "ix_CRF_projected_departure_datetime".

7. [15 pts] Now you have created two indexes. Check the visual query plan again for each of the three queries. Again, you are ***not*** required to submit the visual query plans. However, you ***are*** required to submit the textual "EXPLAIN" output of your queries by following the same steps in Q4.
    a. Query 1)
    b. Query 2)
    c. Query 3)

8. [10 pts] Now, by comparing the query plan of Q4 and Q7, briefly explain how the indexes are being used by the query optimizer to make these queries run faster.

9. [10 pts] If you carefully examine the execution plan of Query 1 and Query 2 after creating the ix_Customer_Email index, you can notice that the index is being used for Query 2. Can you think of a reason why this index is being used?

[Extra Point Questions]

1. [10 pts] Consider the following SQL query.

```
SELECT total_amount, count(*) from DishOrder DO, Dish D, Lounge L
WHERE DO.lid=D.lid and DO.total_amount > 300 and
      D.price > 30 and DO.lid = L.lid and L.airport_IATA_code like 'S%'
GROUP BY DO.total_amount
HAVING count(*) > 1
ORDER BY DO.total_amount;
```

First, check its visual query plan (no need to turn it in). Then find a way to improve its execution speed by creating index(es). To support your argument, attach the "EXPLAIN" output of the query after you create the index(es).

a. The textual "EXPLAIN" plan of the original query.
b. Your statement(s) to create index(es).
c. The textual "EXPLAIN" plan of the query after creating the index(es).