Started on	Wednesday, 28 April 2021, 5:59 PM PDT
State	Finished
Completed on	Thursday, 29 April 2021, 4:45 PM PDT
Time taken	22 hours 45 mins
Grade	**41.0** out of 45.0 (**91**%)

Question 1

Complete

10.0 points out of 10.0

A common data operation in machine learning and statistics is **random sampling.** We divide a large dataset into a training set, validation set, and testing set.

Suppose we want to train a model to **predict flight delays,** oh if only it were that easy. In your midterm database, there is a relation called `airline`.

We are going to divide these airlines into these three groups:

1. 50% of the symbols will be assigned to group 1 (the training set)
2. 25% of the symbols will be assigned to group 2 (the validation set)
3. 25% of the symbols will be assigned to group 3 (the testing set)

Write a query that returns the airline name, and the group (a smallint) that it belongs to. Note that you will need to generate random numbers in order to accomplish this task. There is a standard function that will allow you to do this.

It's very easy to write unwieldy an inefficient queries. To prevent that, you must do the following:

1. You must only make one call to the random number generator.
2. You must minimize the number of subqueries if you use any.
3. Avoid transforming the output of the random number generator.
4. You do not need to compute anything by hand to sketch out the query.

Paste your query into the box.

```
select name,
case
when rand_num < 0.50 then 1::smallint -- training set
when rand_num < 0.75 then 2::smallint -- validation set
else 3::smallint -- testing set
end as group
from (select name, random() rand_num from airline) a;
```

One possible solution. It doesn't transform random() and only uses it once.

```
SELECT
   name,
   CASE
      WHEN rand < 0.5 THEN 1
      WHEN rand < 0.75 THEN 2
      ELSE 3
   END::smallint AS datagroup
FROM (
   SELECT
   name,
   random() as rand
FROM airline
) random_assignments;
```

One student lived dangerously and used RANDOM() twice without a subquery. It works. Try to figure out why:

```
SELECT
  name,
```

```
  CASE
    WHEN random() < .5 THEN 1
    WHEN random() < .5 THEN 2
ELSE 3
END as set
FROM airline;
```

Feedback:

Question **2**

Complete

5.0 points out of 5.0

When writing code, we usually write unit tests to verify that the code we just wrote works properly. We can do something similar with SQL. We can write a query that sanity checks the result of our query.

Write a query that computes the number of airlines that are in each group. Again, we can just treat the group as a smallint. You may notice that we could also compute the percentage, but that further complicates the query, so let's stick with the count. **You should see that approximately 50% of symbols are in group 1, 25% are in group 2, 25% are in group 3. Since we are working with random numbers though, and the number of airlines is small, the result may not be exact.**

```
select b.group,
count(b.group) as count
from (
select name,
case
when rand_num < 0.50 then 1::smallint
when rand_num < 0.75 then 2::smallint
else 3::smallint
end as group
from (select name, random() rand_num from airline) a) b
group by b.group;
```

Possible solution

```
SELECT foo.group AS group
       COUNT(name) AS count
FROM (
   **part 1 query**
     ) foo
GROUP BY foo.group
ORDER BY group;
```

Feedback:

Question **3**

Complete

Not graded

Copy and paste the output from the previous sanity check query below. You can make it look nice by changing the font to Courier.

```
 group | count
-------+-------
     3 |     7
     2 |     7
     1 |    12
(3 rows)
```

Information

In this exercise, we will work with airline flight performance data. This data is from 2001, so you have the opportunity see some airlines that have long since departed (such as HP-America West, US-US Airways and TW-TWA), and others that have not yet been founded (like NK-Spirit). We have four tables for you to work with.

1. **flight** contains information about each flight and several aspects of its departure and arrival
2. **airline** contains a mapping from the IATA code (i.e. UA) to the name of the airline.
3. **airport** contains information about each airport, it's IATA code (i.e. LAX), it's name, city and state, if you need it
4. **state** contains mapping from state abbreviation to state name, if needed.

Question **4**

Complete

2.0 points out of 2.0

Suppose we want to investigate flights whose arrival delays are greater than the airline's average. We write the following query:

```
SELECT
    airline,
    number,
    arrival_delay
FROM flight f
WHERE arrival_delay > (
    SELECT AVG(arrival_delay)
        FROM flight
        WHERE airline = f.airline
);
```

Explain what is conceptually wrong about this query.

This is a correlated subquery that gets the average for each row and then recomputes the table average for each row of the table which is very inefficient essentially about O(n^2) efficiency.

End of lecture 6, start of lecture 7.
This is a correlated subquery. In the subquery we use information (alias) from outside the query. This is very slow because the subquery is recomputed on every row in the outer query. Must be explained and the name correlated subquery must be mentioned.

Feedback:

Question 5

Complete

1.0 points out of 5.0

Write a new query that fixes the issue.

```
select airline,
       number,
       arrival_delay
from flight f
group by airline, number, arrival_delay
having arrival_delay > (
    select AVG(arrival_delay)
    from flight
    where airline = flight.airline
);
```

We did exactly this process at the beginning of lecture 7.

One possible correct answer (must be by airline).

```
SELECT L.airline, number, arrival_delay
FROM flight L
JOIN (
  SELECT airline, AVG(arrival_delay) as avg_delay
  FROM flight
  GROUP BY airline
) sq
ON L.airline = sq.airline
WHERE L.arrival_delay > sq.avg_delay;
```

Feedback:
-4 This is still a correlated subquery, though it's called once per group instead of once per row, it doesn't fix the original problem

Question 6

Complete

5.0 points out of 5.0

Write a query that computes the number of flights flown for each origin/destination pair. Sort the output so that the most frequent route is the first result.

```
select origin,
destination,
count(*) as cnt_flights
from flight
group by origin, destination
order by cnt_flights desc;
```

Very similar to HW 1/2. See lecture 5.

One possible solution. Few others were accepted due to the simplicity of the query.

```
SELECT origin, destination, COUNT(*) as total_flights
FROM flight
GROUP BY origin, destination
ORDER BY total_flights DESC;
```

Feedback:

Question 7

Complete

8.0 points out of 8.0

Write a query to compute the average flight delay (in hours) by airline. Since many airlines went out of business during this time, **restrict the result to only contain airlines that saw 6,000 or ore flights in the dataset and exclude canceled flights.** There is a column in the `flight` table called `arrival_delay`. Note that it is a simple integer, which should help you greatly. We are ignoring departure delay and assuming that passengers only care about how delayed they are getting to their location. There is also a column called `cancelled` that should help you a lot.

Your query should output the airline **name** (not code) and the **average arrival delay (in that order) and also sort the rows in descending order by average delay.**

Some other things and a summary:

1. You should not need a subquery here.
2. Output the airline **name** and the average arrival_delay.
3. Only report airlines that have completed at least 6,000 flights in the year.
4. Exclude any canceled flights.

```
select name,
avg(arrival_delay) as arrival_delay
from flight
join airline as a
on airline = code
where cancelled != true
group by name
having count(*) > 6000
order by arrival_delay desc;
```

See lectures 5 and 6. We did a very similar example.

One possible correct response: (very few other answers were accepted)

```
SELECT L.name, AVG(arrival_delay)
  FROM flight L
  JOIN airline R
  ON L.airline = R.code
  WHERE L.cancelled = 'f'
GROUP BY R.name
HAVING COUNT(flight.airline) >= 6000;
ORDER BY AVG(arrival_delay) DESC;
```

Feedback:

Question **8**

Complete

10.0 points out of 10.0

The table **flight** contains a sample of flights for various airlines in the United States in 2001.

Write a query that computes the number of flights flown by each airline in the past 7 days (inclusive), and sort the output by airline (just the code is fine) and the date, both ascending and of course print the count. Note that this must be computed for every arbitrary date, it's a not a weekly Sunday-to-Saturday computation. There are date gaps -- so just use the last 7 calendar days, not the last 7 days *that are in the table*.

Hint 0: To start off, fix a particular date and find all rows within the past 7 days (including the fixed date), ignore the time of day. Then, with those, you will compute the total number of flights by airline over that subset of the rows. By "fixed date" I mean the following. Suppose the row we are looking at has a date of 2001-04-30. To compute the 7 day number of flights at 2001-04-30, we retrieve all of the rows from 2001-04-24 to 2001-04-30 inclusive. We compute the total number of flights over that window, then we move to 2001-05-01, do the same, and so on.

Hint 1: This can be solved using methods we learned in class. If you use a different method and do it incorrectly, you will not receive credit.
Hint 2: You may need to use the documentation to find the proper function(s) to use for part of the query. If you get stuck, take a look at the schema for this table in midterm.sql. The creator of the data used much simpler date and time types and this should actually help you.
Hint 3: Be very careful with your aliases.
Hint 4: There is no subquery in this problem.
Hint 5: Be very careful about your join conditions. Accidentally missing one leads to a CROSS JOIN.

```
select distinct a.airline, a.flight_date, count(*) as cnt
from flight as a
join flight as b
on a.airline = b.airline
and b.flight_date between a.flight_date - 6 and a.flight_date
group by a.airline, a.flight_date, a.id
order by a.airline, a.flight_date asc;
```

We did a very, very similar example in lecture 6. See the playing cards example.

One possible correct response:

```
SELECT
  DISTINCT
  L.airline,
  L.flight_date,
  COUNT(R.id)
FROM flight L
JOIN flight R
ON L.airline = R.airline AND L.flight_date - R.flight_date BETWEEN 0 AND 6
GROUP BY L.flight_date, L.airline, L.id
order by L.airline, L.flight_date;
```

If L.id is not in the GROUP BY, must use COUNT DISTINCT in the SELECT.

Feedback:

◄ CS143 Spring 2021 Midterm, ...

Jump to...

Lecture 10 Slides (1PP) ►