Hi friend, this isn't the best pdf because I used Ctrl+P on CCLE, which means that some questions are cut off and there is cursed text in places. Just ignore the cursed text, and try your best to understand. The brown letters are the answers to the questions.

Started on	Wednesday, 29 April 2020, 5:00 PM PDT
State	Finished
Completed on	Wednesday, 29 April 2020, 6:05 PM PDT
Time taken	1 hour 4 mins
Grade	**45.0** out of 55.0 (**82**%)

Question **1**

Complete

2.0 points out of 2.0

Take a look at this table from Lecture 3. Fill in the blanks such that {M, Z} is a **superkey (composite), but such that *B* and *D* are not *candidate keys* for this table.**

EmployeeID (A)	EmployeeName (B)	SSN (C)	ManagerID (D)
1	John Smith	123456789	9
2	Sarah Decker	242424246	3
3		987654321	
4	Jane Allison	112348132	1

Value for *B* in the incomplete row:

John Smith

Value for *D* in the incomplete row:

3

(There are also many other values that work)

Several answers accepted here. Here are two examples:

B=John Smith
D=1 or 3

B and D are still a superkey, but B and D themselves are not candidate keys.

B=Jane Allison
D=3 or 9

Question **2**

Complete

2.0 points out of 2.0

Briefly explain why relational algebra operators must remove duplicates.

Relational algebra operators must remove duplicates because they deal with sets, and in sets there are no duplicates.

Relational algebra operates on **sets** of tuples. If we allowed duplicates, we would have a multiset/

Complete

5.0 points out of 8.0

contains Y tuples. Assume that $w > 0$, $Y > 0$. Make no assumptions about keys. For each of the following expressions, state in terms of w and Y the minimum and maximum number of tuples that could be in the result of the expression.
Note: The correct response may require a function (min, max, etc.) to be applied to w or Y.

Due to CCLE limitations, all parts of this problem are listed below. **Please double check that you have written a response for all parts, before submitting.**

(a) $\Pi_{A,\,C} R \bowtie S$

(b) $S \bowtie (S \bowtie S)$

(c) đ RIGHT OUTER JOIN Ď

(d) $\Pi_B(R) - (\Pi_B(R) - \Pi_B(S))$

(a)

Minimum: 0

Maximum: r * s

(b)

Minimum: s

Maximum: s^3

(c)

Minimum: s

Maximum: r * s

(d)

Minimum: min(r, r - (r - s))

Maximum: max(r, r - (r - s))

-1 for each error.

(a) Min: **0** Max: **rs**

(b) Min: **s**, Max: **s**

(c) Min: **s**, Max: **r * s**

(d) Min: **0** Max: **min(r, s)**

Complete

6.0 points out of 8.0

are no duplicates in this table **`friends`**.

Your task is to identify each line that contains an error and state what the error is. Use the numbered lines in the response template to write your response. Note that many lines may need to be left blank if there is no error on those lines.

The query is below. Before getting carried away, there are no column aliasing issues, most lines with errors only contain one error. Many lines do not contain any errors.

NOTE: The context isn't important, but this is a variation of the FOAF problem, except now we have added another column, `status`, that labels a student as an Undergraduate or Graduate. We want to compute the average number of FOAFs undergraduates have, and the average number of FOAFs that graduate students have.

```
1  SELECT
2      uid,
3      status,
4      AVG(number_of_foafs) AS average
5  FROM (
6      SELECT
7          uid,
8          status,
9          COUNT(friend_uid) AS number_of_foafs
10     FROM (
11         SELECT
12             L.uid,
13             IFELSE(L.status=='U', 'Undergrad', 'Graduate') AS status,
14             R.friend_uid AS friend_uid
15         FROM friends
16         SELF JOIN friends
17         ON L.friend_uid = R.uid AND L.uid != R.friend_uid
18     )
19     GROUP BY uid, status
20     WHERE number_of_foafs > 2
21 )
22 GROUP BY status
23 WHERE uid != NULL;
```

Your Responses

Line Number Errors, if any. Leave the cell blank if there is no error on that line.

1	
2	GROUP BY will complain about uid because it is not being used in aggregations
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	I don't think IFELSE is an actual function
14	
15	
16	I also don't think SELF JOIN is an actual term
17	
18	
19	GROUP BY must be after WHERE
20	aliases created in SELECT cannot be used by WHERE: number_of_foafs is used
21	
22	GROUP BY must be after WHERE
23	Cannot compare to NULLs with simple equality operators; use IS NOT NULL instead

13. IFELSE doesn't exist (use CASE). [== should be =]
15. Missing table alias (note that the hint says **column** aliases are correct)
16. No such thing as SELF JOIN, missing table alias L
19/20. GROUP BY and WHERE in wrong order.
20. HAVING not WHERE. Cannot use number_of_foafs.
22/23. GROUP BY / WHERE in wrong order
23. Use IS NOT NULL not != NULL
2. uid should be deleted.

Question 5
Complete
1.0 points out of 2.0

List two ways to prevent SQL injections.

Method 1:
Use prepared statements
Method 2:
Use cursors to execute the query

Or just don't use string concatenation/building to construct a query given the raw input of a user of the application.

1. Use a prepared statement/query
2. Proper authorization on the table.

Question 6
Complete
5.0 points out of 5.0

Suppose we have a relation called `cs143_midterm` containing student information and the score each student received on this here midterm. The relation is pictured below. This comes from the lecture slides.

uid	last	first	mi	midterm	major	level	career
246802468	SCHMOE	JOE		58	SOCIOLOGY	UFR	UG
123456789	BRUIN	JOSIE		98	MATH	GD2	GD
012345678	BRUIN	JOE		56	ECONOMICS	UJR	UG
999999999	FRANCO	JAMES		65	ENGLISH	USR	UG
111111111	BEAR	THEODORE	E.	20	MATH	GMT	GD
987654321	COYOTE	WILEY	E.	81	SOCIOLOGY	UJR	UG
121212121	BLOCK	GENE	D.		BIOLOGY	CHN	CH

Below we have an innocent query that simply retrieves your own name and midterm score when you enter your UID. Write a SQL injection using this query that will **change** Joe Bruin's score (suppose you are Joe Bruin) to 100. It is safe to assume that the public can read/write this table. Your attack query must **only** change your midterm score, and not return any data.

Be very careful. It's very easy to accidentally construct a "safe" query. We recommend that you copy/paste the existing query and then add your injection into it, rather than just specifying the injection directly.

```
"""
SELECT
    uid, last, first, mi
FROM cs143_midterm
WHERE uid=
"""
```

```
SELECT
  uid, last, first, mi
FROM cs143_midterm
WHERE uid='' AND 1=2; UPDATE cs143_midterm SET midterm = 100 WHERE uid = '012345678';
```

```
SELECT
  uid, last, first, mi
FROM cs143_midterm
WHERE uid='' AND 1=2; UPDATE cs134_midterm SET midterm=100 WHERE uid='012345678';
```

The AND 1=2 is crucial, otherwise we return data.

Complete

1.0 points out of 3.0

Because standard views exist, and if someone wants to waste the resources just to update the data in a materialized view, they might as well just use a standard view and use less resources.

A materialized view is similar to a temporary table. Updating this materialized view will update this temporary table, but there is NO relationship between the materialized view and table it was derived from. So, updating a materialized view is a no op, and will actually give an error in Postgres.

A materialized view is cached to disk and really has no relationship to the original table that it was created from. Because of that, updating a materialized view is useless. It does not update the underlying table. The RDBMS will not allow it in most cases, and even if it did, it would be a no op.

Question 8

Complete

2.0 points out of 3.0

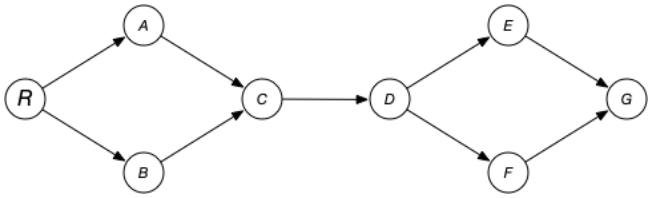
We learned that an updatable view can only be defined on some very basic queries. An updatable view cannot be defined using a query that includes a GROUP BY. **Briefly explain why not.**

GROUP BY queries represent aggregations of data, so when you try to insert data through an updatable view it does not make sense with the context of the aggregations; you might not be seeing an accurate representation of the underlying table.

If we update a view containing a GROUP BY, we would be adding an **aggregate** into the underlying table, which does not make sense. The underlying table records individual rows, not aggregates.

Information

DIRECTIONS: Use the authorization graph provided below to answer each question. đ represents some root user.



In multiple choice problems, if none of the answers are true, you must explicitly choose the proper "None of the Above" option. Leaving all of the options unmarked will result in 0 credit.

Question 9

Correct

3.0 points out of 3.0

Which users **were granted** the privilege ỵ using the `WITH GRANT OPTION`?

Select one or more:

- [x] !
- [x] M
- [x] Q
- [x] Z
- [x] c
- [x] à
- [ ] ä
- [ ] None of Them

Correct

3.0 points out of 3.0

Select one or more:

- [ ] !
- [ ] M
- [ ] Q
- [x] Z ✔
- [x] c ✔
- [x] à ✔
- [x] ä ✔
- [ ] None of Them

Question **11**

Correct

2.0 points out of 2.0

We can directly revoke the privilege ỵ from Q. We can also ǫŎKǫƠùĂǻ revoke the privilege ỵ from Q via **CASCADE** by instead revoking the privilege ỵ from...

Select one or more:

- [ ] đ only
- [ ] ! only
- [ ] Monly
- [ ] Either ! or M
- [x] Both ! and M ✔
- [ ] None of These Choices is Correct

We accepted two answers here. The intention was for you to identify nodes that would allow us to remove the privilege from Q. Thus, **Only R** and **Both A and B** would be correct. **R** can revoke the privilege from C. **A and B** together must revoke the privilege for C to lose it.

We also accepted **A and B** because the question states ǫŎKǫƠùĂǻ and the word "from" convinced some students we were asking about revoking the privilege from the root, which isn't allowed.

Question **12**

Correct

2.0 points out of 2.0

If we directly revoke the privilege ỵ from Q using **RESTRICT**, which role(s) lose the privilege?

Select one or more:

- [ ] !
- [ ] M
- [ ] Q
- [ ] Z
- [ ] c
- [ ] à
- [ ] ä
- [x] None of Them ✔

Question **13**

Correct

2.0 points out of 2.0

Suppose Q is a bad guy, and ! grants him the privilege ỵ **WITH GRANT OPTION**. He knows that eventually it is going to be revoked so he grants ỵ to his friend M**WITH GRANT OPTION** so that Mcan restore the privilege to C after it is revoked from him. **Write a brief explanation of why this plan may not work.**

Answer: If we revoke with CASCADE from C, then B will also lose the privilege of granting the ✔

This would not work if revoke uses the CASCADE option. When Q loses the privilege, so does B.

Complete

3.0 points out of 3.0

application.

This requires the application to maintain the connection to the database, which is then a waste of resources.

Several possible answers.

1. We have to execute a new query each time. SLOW. This was the most common answer.
2. Data may change in the underlying table as we are querying.
3. If we use a cursor instead, we must maintain an open connection and transaction to the DB which consumes resources.

Using a cursor is a much better approach, but there are still performance considerations to take into account.

Question **15**

Complete

3.0 points out of 4.0

You are working for a startup and want to create your first relational schema! Your coworker says:

"Let's not spend too much time worrying about the schema. Just use the largest data types for each column, we can always make them smaller later. Also, don't worry about the number of columns we have. If we need to add any columns later, we can do so with an `ALTER TABLE`."

Identify and explain all of the flaws in your coworker's thinking. For each flaw, comment on what we should do instead. Be specific, but concise.

1. "We can always make them smaller later." This reasoning is very bad as it may cause the information in the big data types to be lost when they are shrunk. This can be disastrous for primary keys especially.

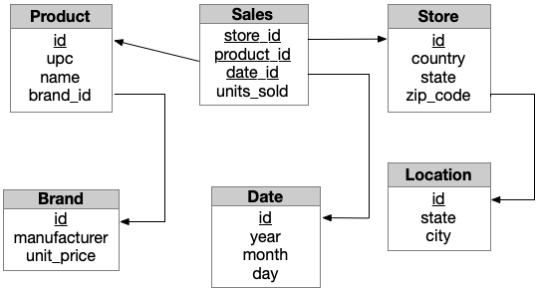
2. "[illegible] This can potentially break any code that previously relied on the presence of less columns. Apparently, it may also add JSON blobs.

1. We should spend considerable time worrying about the schema.
2. Using the largest types is wasteful. Start with the smallest types necessary and then promote when needed.
3. We NEVER want to demote the column type or we may lose data.
4. We shouldn't add columns later.
5. Instead, we should have an extra column and store data into it as a JSON or XML blob. (did not penalize if missing)

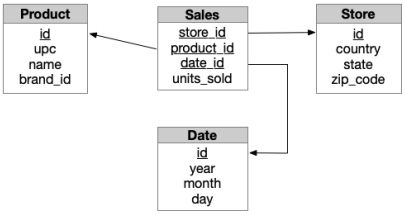
Correct

3.0 points out of 3.0

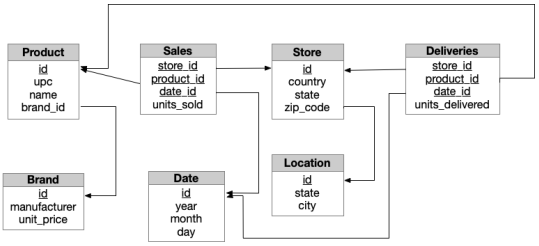
based on these diagrams, we leave it to you to determine which is which. **Hint:** Focus on the ÁıẏGY of tables and how they are related and Ör Âjust the shape in the diagram.



Snowflake ✔



Star ✔



Constellation ✔

Started on	Wednesday, 29 April 2020, 6:07 PM PDT
State	Finished
Completed on	Thursday, 30 April 2020, 5:57 PM PDT
Time taken	23 hours 50 mins
Grade	**43.0** out of 45.0 (**96**%)

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 the stock market,** oh if only it were that easy. In your midterm database, there is a relation called **`stock_symbol`.**

We are going to divide these stock symbols into these three groups:

1. 50% of the symbols will be used for training (call it group 1)
2. 30% of the symbols will be used for validation (call it group 2)
3. 20% of the symbols will be used for testing (call it group 3)

Write a query that returns the `symbol`, and the model 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, but be careful how you use it. Then, you can partition these random numbers according to the schema above.

```
--I could not think of a way to avoid the multiple case statements
SELECT
   symbol,
   (CASE floor(random()*10)+1
      WHEN 1 THEN 3
      WHEN 2 THEN 3
      WHEN 3 THEN 2
      WHEN 4 THEN 2
      WHEN 5 THEN 2
      ELSE 1
   END)::smallint model_group
FROM stock_symbol;
```

Comment:

```
floor(random()*10)+1 is CEILING.
```

Question **2**

Complete

3.0 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 stock symbols 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, 30% are in group 2, 20% are in group 3. Since we are working with random numbers though, the result may be off by up to 5%.**

```
SELECT
   model_group,
   COUNT(symbol) num_syms
FROM problem 1            -- replace with results from problem 1
GROUP BY model_group;
```

Comment: Purpose problem was to use a subquery. COUNT / GROUP BY correct. -2

Complete

Not graded

to Courier.

```
 model_group | num_syms
-------------+----------
           3 |      106
           2 |      140
           1 |      259
(3 rows)
```

```
-- This result is from when I use problem 1's query as a subquery in problem 2's FROM clause
```

```
-- Which also means that it varies each time due to random()
```

Information

In this exercise, we will work with stock prices. We have three tables for you to work with.

1. **`listings`** contains information about each stock symbol from the S&P 500, such as company and industry.
2. **`stocks`** contains pricing data for each stock for 5 years (2013-2018). It includes
   - date: the market date
   - open: the price that the stock was at when the market opened that day
   - close: the price that the stock was at when the market closed that day
   - high: the highest price that the stock traded at during that day
   - low: the lowest price that the stock traded at during that day.
   - symbol: the stock symbol (e.g. **`LUV`** for Southwest Airlines, **`GOOG`** for Alphabet)
   - There are other columns we will not use.
3. **`southwest`** contains stock data for LUV (Southwest Airlines) only, for 5 years. It has the same schema as **`stocks`**.

Question **4**

Complete

5.0 points out of 5.0

Before we can do anything, we should explore the data a bit. This is an important step. Making assumptions about the data can cause a lot of problems.

One assumption is that the first row in the table comes first chronologically, and that the last row comes last chronologically. If this data was combined from multiple systems, it's possible the data is mixed.

Write a query that finds the earliest date in the `stocks` data as well as the latest date in the `stocks` data. Return one row containing both.

```
SELECT
    MIN("date") earliest,
    MAX("date") latest
FROM stocks;
```

Comment:
Don't need the quotes

Complete

5.0 points out of 5.0

query should return the stock symbol, the average open, close, high and low prices over the entire period. Give your resulting columns nice names.

```
SELECT
    symbol,
    AVG(open) avg_open,
    AVG(close) avg_close,
    AVG(high) avg_high,
    AVG(low) avg_low
FROM stocks
GROUP BY symbol
HAVING AVG(close) < 20;
```

Comment:

Question 6

Complete

10.0 points out of 10.0

The table `listings` contains information about each stock symbol, such as the company name and the industry it belongs to.

For each stock symbol and each trading day, we can compute the difference: `high - low`. We can then summarize this over the entire data window (5 years).

Write a query that computes, for each stock symbol in the Energy sector, the *largest* difference between the high and low price over the entire period. The query should return the stock symbol, the company that the stock symbol refers to, and this largest difference. Sort from largest to smallest.

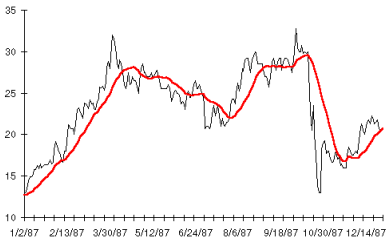
```
SELECT
    stocks.symbol,
    name AS company_name,
    MAX(high - low) largest_diff
FROM stocks
JOIN listings
ON stocks.symbol = listings.symbol
WHERE sector = 'Energy'
GROUP BY stocks.symbol, name
ORDER BY largest_diff DESC;
```

Comment:

Complete

10.0 points out of 10.0

In time series modeling (which stock modeling typically involves), we can apply smoothing to remove day-to-day variations in stock price. For some context, the graphic below shows the stock price for Autodesk Inc. in 1987. The **black** line is the individual stock price and the **red** line is the moving average (smoothing) over what seems to be 7 days.



Write a query that computes the 30 day moving average for the following 4 metrics for LUV and sort the result from earliest date to latest date:

1. opening price (open)
2. closing price (close)
3. high price (high)
4. low price (low)

To start off, fix a particular date and find all rows within the past 30 days (including the fixed date). Then, with those, you will compute the average prices over that subset of the rows. By "fixed date" I mean the following. Suppose the row we are looking at has a date of 2015-04-30. To compute the 30 day moving average for 2015-04-30, we retrieve all of the rows from 2015-04-01 to 2015-04-30. We compute the average over that window, then we move to 2015-05-01, do the same, and so on.

Hint 1: If you get stuck on how to approach this problem, look in HW 2 and 3.
Hint 2: You will need to use the documentation to find the proper function(s) to use for part of the query.
Hint 3: Be very careful with your aliases.
Hint 4: There is no subquery in this problem.

```
SELECT
    curr.date,
    AVG(past.open) avg_open,
    AVG(past.close) avg_close,
    AVG(past.high) avg_high,
    AVG(past.low) avg_low
FROM southwest curr
JOIN southwest past
ON curr.date - past.date BETWEEN 0 AND 29
GROUP BY curr.date
ORDER BY curr.date;
```

```
--throwback to those 5 minutes when prof talked about window functions and non-equijoins!!!
```

Comment: