CSCI 585, midterm exam, 6/11/20

Please read the following carefully, before starting the test.

• the exam is open books/notes/devices - feel free to look up whatever you want!

• there are 7 questions plus a 'non-data-related' bonus, for a max of 35 points

• there are no 'trick' questions, or ones with long calculations or formulae

• please do NOT cheat - this means NOT communicating with anyone via any device/medium/channel - you will get a 0, and be reported to SJACS, if you are found to have cheated; ANY attempt to get help from others in any form is a VIOLATION, as per https://policy.usc.edu/scampus-part-b/, sections 11.11 through 11.14 [read it, if you are not familiar with it]

• when the time is up (75 minutes), stop your work, then spend the rest of time (30 minutes) on submission

Good luck!

Q0 [0 points]. DO turn this in - DO NOT omit doing so.

Please write the following line, and sign it - it is your acknowledgment of having read USC's policies on academic misconduct (https://policy.usc.edu/scampus-part-b/, 11.11-11.14) and agreement to honor them.

I have read USC's standards on academic integrity, and agree to abide by them.

Q1 [2*2 = 4 points].

'Loose coupling/loose dependence is preferable to tight coupling/tight dependence.'

What are TWO different cases where the above is true, in the world of data-handling (ie. in what we've covered in the course)?

Answer:

1. File system DBs vs other kinds, eg. relational DBs. In a DB, we want the DB engine for adding, modifying, deleting and querying data, to be de-coupled from the internals of how the data is stored, ie we want structural independence.

2. With DB connectivity, we'd like loose coupling between data consumers (eg client apps that use web services), and data producers (the systems that supply data, in response to service/microservice requests). In other words, an answer could be services/microservices, where requests and responses are decoupled.

For each case:
+2 for correct answer which is very similar to this
~~-1 if decoupling between system and data is not mentioned~~
[the correct answer would need to involve an example where decoupling is desirable, like in the two cases I've shown].
+1.5 partial credit if some other case is written which is very similar to the correct answer

Q2 [3*2 = 6 points].

We saw examples of where a SQL query was intermixed with C#, Java, Python... **What are practical reasons** (at least two) for needing to do so?

Answer:

In real-life, most data is accessed via apps and other client software, that are written in host languages such as C++, Java, Python, JS etc, that contain the UI code, presentation logic, etc. So there needs to be way for these apps to communicate with a DB engine that expects SQL queries - so there is intermingling, typically in the form of SQL queries being handled as strings in the host language and handed off to the SQL engine.

Non-SQL languages have powerful data structures and associated methods, and language-level features (eg functional programming, iterators etc) - the programmer is able to leverage these, by mixing them with SQL (eg. resultsets returned from the DB engine can be traversed using methods such as .next()).

(+3 for each correct answer which is very similar to this)
(+2 partial credit if some other reason is written which is very similar to the correct answer)
(0 if the reason is not practical and/or doesn't make sense)

Q3 [5 points].

An investment company has invested its members' wealth, in a variety of holdings: stocks, real-estate, gold and other forms of jewelry, antiques, famous paintings. You are asked to help catalog the assets. How would you represent the wealth being invested, via a simple EER diagram? You can assume whatever you need (in regards to representing the various types of holdings).

Answer:

Multiple answer variations are possible - overall, a simple two level design will do.

Superclass entity: Asset, with these columns: AssetID (PK), MemberID, PurchaseDate, PurchaseAmt, TodaysWorth, AssetType [subtype discrim column, MUST be disjoint]

Member (MemberID, Name, Address, Phone, Email, MemberSince, MemberClass, etc): links to Asset as 1:M

Subclass for each type of asset, eg. Stock, RealEstate, Valuables, Antiques, Painting [each will have specific columns, eg. for Painting: Medium, Size, Painter, YearPainted...].

Rubrics:
-0.5: If incorrect relationship between member and asset.
-0.5: If incorrect relationship between superclass and subclass (disjoint)
-1: If member entity not mentioned
-1: If asset is not the Superclass entity
-1: If only some of the subclass are mentioned
-2: If none of the subclass are properly mentioned
-1: If all attributes are not mentioned
-0.5: If some attributes are mentioned

Q4 [3+2 = 5 points].

What is the problem with the 'original' CAP Theorem? Explain in your own words, in a few sentences. **What is the** modern, preferable alternative formulation? Do not simply state the CAP Theorem - answer the questions asked!

Answer:

In the original CAP theorem, partition tolerance ('P') was on equal footing with C and A, making it seem like there was a choice between CA, AP and PC that a DB designer could provide end-users; but in reality, CA is never a choice (dropping P is not an option).

In the modern interpretation, we consider the theorem as providing a choice between availability vs consistency, when a partition failure occurs - whether we keep operating (A) while letting data become inconsistent, or prioritize consistency at the expense of lowering system availability.

Rubrics:
Part 1 (3 points): Include CA is not realistic / low P is not a option, and give correct explanation gets full marks
-1: if specify CA not realistic but the explanation is wrong.
-2: if only specify we can choose at most 2 out of 3 [CA, PA, PC] or cannot achieve CAP all at once
-3: if totally wrong or unrelated

Part 2 (2 points): Include choice between A and C gets full marks, else include BASE or PACELC also gets full marks, or if they state the principle of BASE of "sacrifice consistency in favor of availability" .
-2: if not related to above

Q5 [2+3 = 5 points].

What does the following query do?

```
SELECT x.Name,
       x.City,
       (SELECT CompanyName FROM Company WHERE CompanyID =
x.CompanyID) AS CompanyName
FROM Customer x
```

The query is better expressed as follows. **Why?**

```
SELECT r.Name,
       r.City,
       c.CompanyName
FROM Customer r
    LEFT JOIN Company c
         ON r.CompanyID = c.CompanyID
```

Answer:

Given a Customer and Company table, the query lists the name, city, company (including null) for each customer.

The join query is better, because the joining is a one-time operation that produces the same result - but in the previous case, the correlated query required the Company table to be scanned repeatedly, for each customer in the Customer table (highly inefficient!).

Rubrics:
Part 1:
2points, if the answer is written correctly or as close to one above .
Either full or zero

Part 2:
2 points for specifying why the join query is better
1 point for specifying previous query as correlated query(and its inefficiency).

Q6 [1*6 = 6 points].

How does normalizing tables, help or hinder the following?

• data integrity

• querying

• creating and using indexes

• data updates

• concurrency

• DB design

Answer:

Integrity: Goes up, because data is kept in just a single location, and referenced from it elsewhere.

Querying: can be easy if searching on one or a small # of tables, but with a large number of participating tables, join queries are more verbose, and are slower (inefficient).

Indexes: Easier to create, and more importantly, be used by the query engine. It helps in faster data retrieval.

Updates: Easy and fast, since we only update a portion of our entire DB [including the fact that we only need to update fewer indexes]

Concurrency: Also goes up, since locking is restricted to just the tables/rows where there is contention (multiple requests).

DB design: Helps create a clean design with no partial or transitive dependencies. There must be balance in normalization and performance. Overly normalized DB will not perform optimally

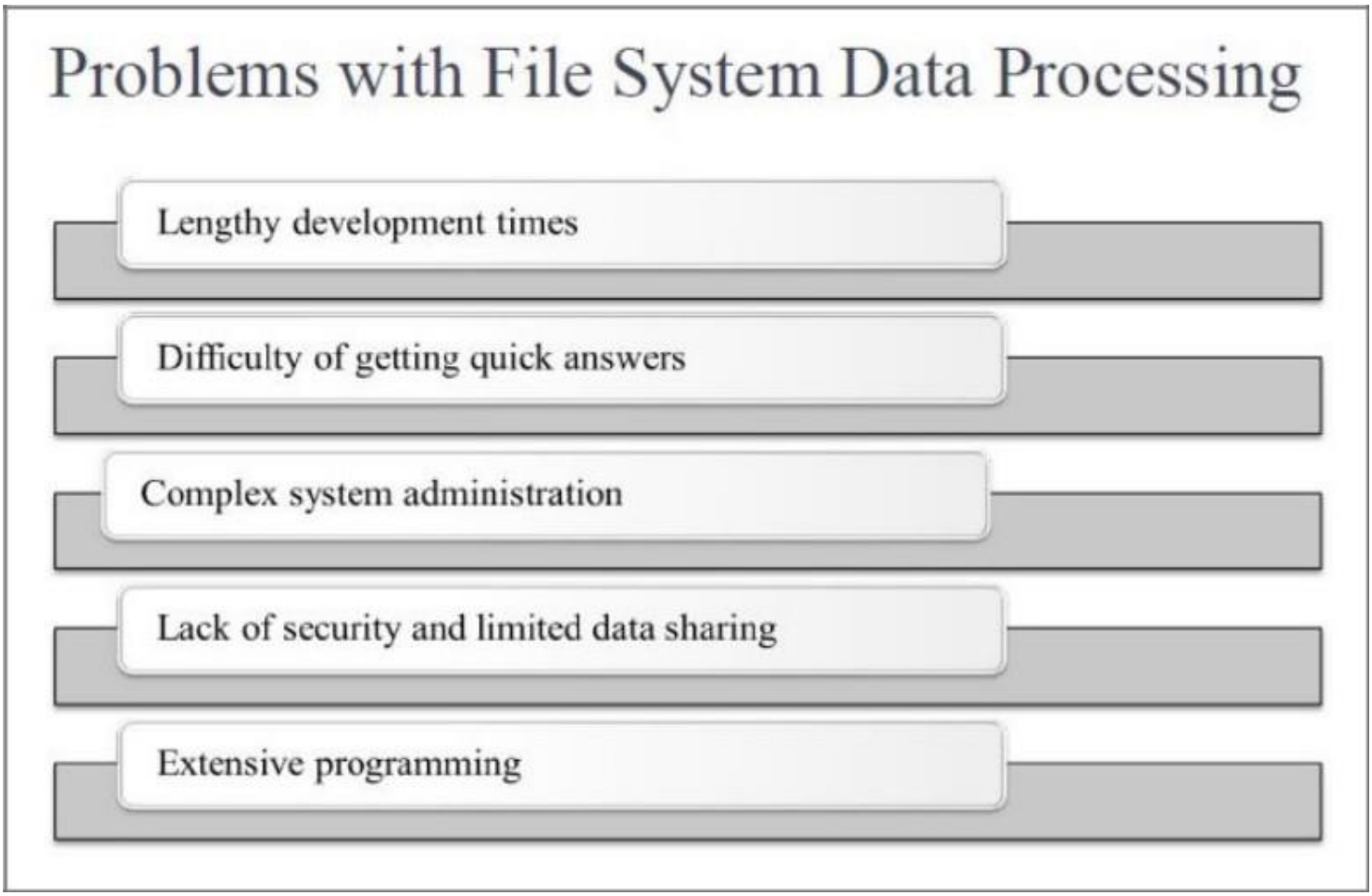
Rubrics:

1 point for each of the above points if mentioned correctly about how normalization helps/prevents in providing integrity,querying etc

-0.5 if a point doesn't specify correctly how does it assists/prevents
-0.5 if a student writes normalization will always help in DB design. Overly normalized DB will not perform optimally

Q7 [2*2 = 4 points].

We discussed the following problems, when it comes to using a simple file system as a database:



The above problems are all due to a single core reason: lack of a

standard way to query the data. **What are additional problems** (at least two) with using loose files to store and query data?

Answer:

1. Lack of security and access restrictions (harder to enforce).

2. No way to enforce transaction management (eg two-phase locking) for concurrent transactions.

3. Difficulty in updation because of structural dependence.

4. Data Redundancy and the associated problems like data inconsistency.

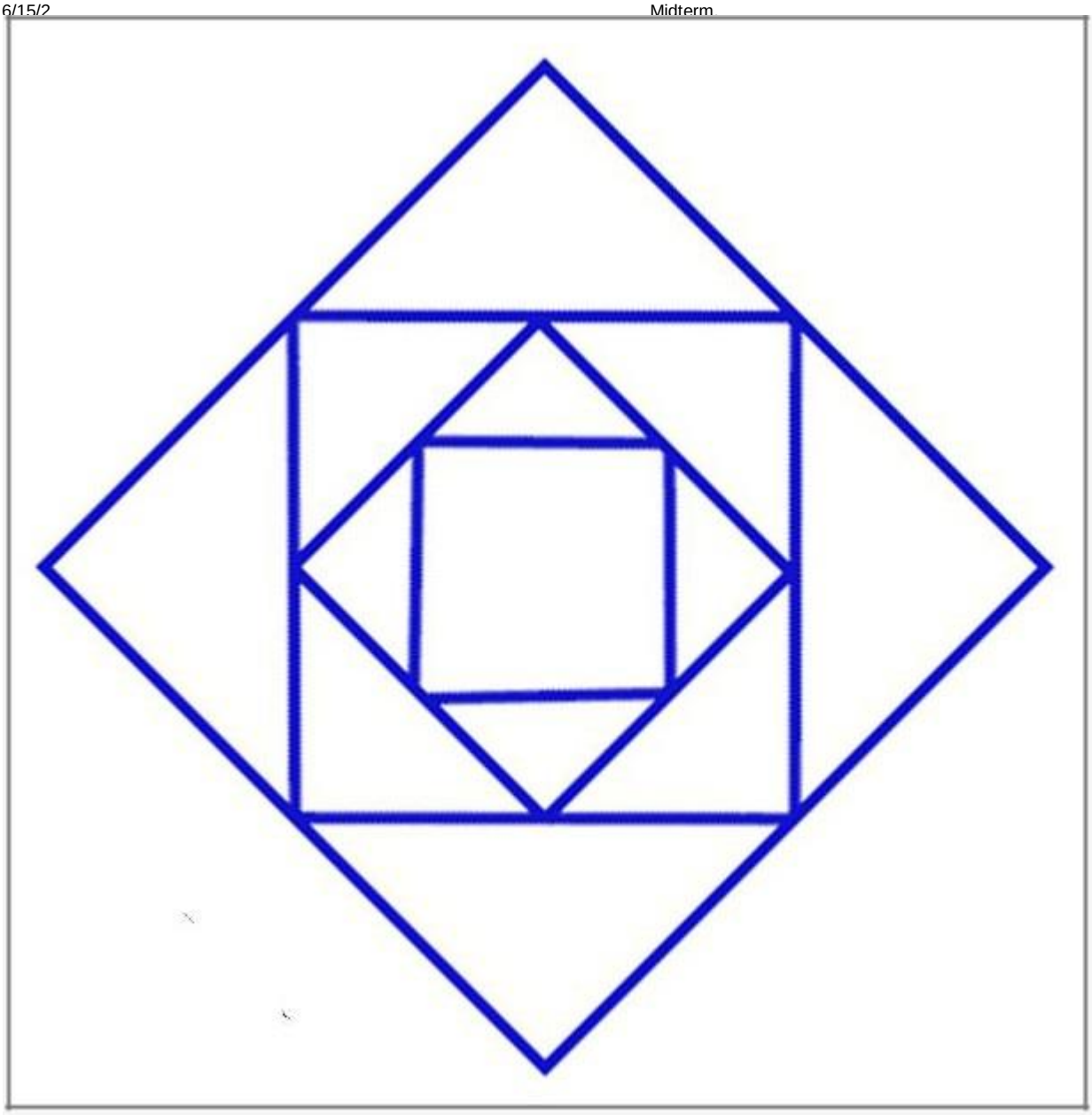
Rubric:

2 points each for any of the two points mentioned above or Any answer that points to RDBMS features (that are absent in file system DBs) is acceptable.

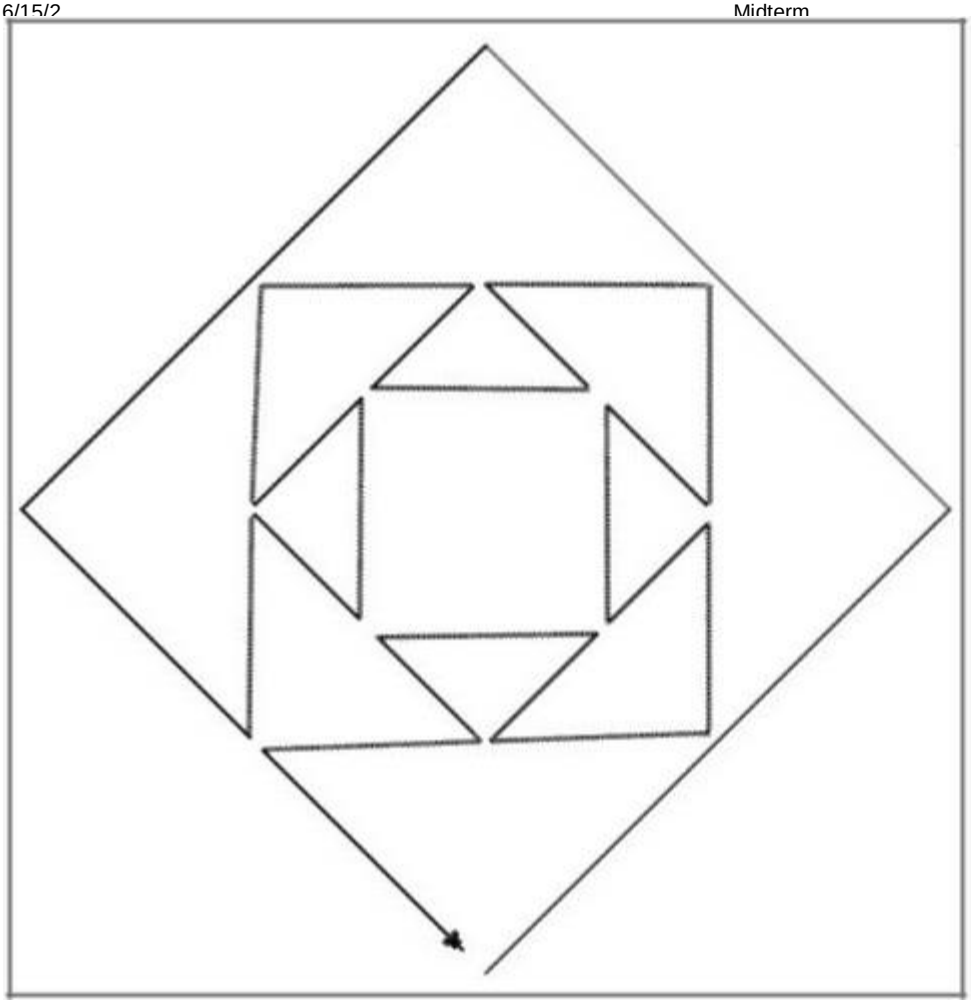
-2 if one of the problem doesn't point out to RDBMS features -1
if a problem partially mentions about the problems
0 if both (or more problems) don't mention the RDBMS features.

Bonus [1 point].

How would you draw the following blue figure using just a single, unbroken line where you don't draw over what you already drew? In other words, you can't lift the pen while drawing, and, you can't draw over even a part of an existing line.



Answer:



Rubric:
Full 1 mark or 0 for drawing the above figure correctly.