CSCI 585 Midterm Exam Fall 2020

Sample Solutions
&
Rubrics

Q0 [0 points]. DO turn this in - DO NOT omit doing so [you will LOSE 5 points if you skip this].

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/ (https://policy.usc.edu/scampuspart-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 [2.5*2 = 5 points].

Modern applications that run on the Internet are able to connect to multiple databases, via microservices. In the early days of databases, applications were monolithic and centralized, in 'SDSP' fashion. Trace the evolution, from the old to the new - in other words, what were the intermediate steps? Discuss at least 2 steps.

A. A variety of intermediate steps exist(ed) that can be named+discussed: ODBC, JDBC, COM, COM+, CORBA, cgi-bin (server-side scripts), client-side support (eg. using ActiveX, Node...), dedicated web-to-db middleware.

Rubrics:

2 or more technologies mentioned and described from above list fetch 2.5 * 2 = 5 points total.

1 point for mentioning technology name

1.5 points for appropriate technology description

Q2 [2+4 = 6 points].

A typical DB transaction consists of table reads and writes in a speci c order - the order matters, because earlier operations typically a ect later ones (eg we read from a cell value, multiply it by 1.1, write it into the cell). In an interleaved transaction schedule, multiple transactions' reads and writes are included, preserving each transaction's order of reads and writes. The DBMS allows the schedule to proceed, as long as the schedule is 'serializable', ie. as long as it is equivalent to the transactions running serially (in sequence, one a er another). But if the schedule is not serializable, one or more transactions would need to be aborted and restarted.

2a. Why not make all schedules always run serially?

A. Doing so will make the throughput (processing rate) su er, ie. users would face longer wait times compared to running transactions interleaved.

To check if a schedule is serializable (and therefore executable by the DBMS), we can SWAP two non-con icting operations at a time, multiple times, till we transform it to a serial schedule (note that two

operations con ict if they belong to di erent transactions, operate on the same cell value, and one or both of them write to the cell).

2b. Is the following schedule, serializable or not, and why (show your steps)? In the schedule shown, A and B are the data items (cells), R and W are read and write operations, transactions are numbered 1 and 2.

```
R1(A) R2(A) R2(B) W2(B) R1(B) W1(A)
```

A. We can analyze the schedule in one of two ways - by swapping noncon icting pairs of operations like mentioned above, or, by 'sliding' them visually along the timeline, to group together steps belonging to each transaction [which amounts to the same thing as the rst technique, but the sliding needs to respect con icting ops, ie. should produce the SAME serial result].

Swapping:

```

```

```
R2(A) R1(A) R2(B) W2(B) R1(B) W1(A)
[swap R1(A),R2(A)]
```

```
R2(A) R2(B) R1(A) W2(B) R1(B) W1(A)
[swap R1(A),R2(B)]
R2(A) R2(B) W2(B) R1(A) R1(B) W1(A)
[swap R1(A),W2(B)]
```

Since swapping yielded transaction 2 followed by transaction 1, we conclude that the original schedule is indeed serializable.

On a timeline [where time increases downwards]:

```
     R2(A)
     R2(B)
     W2(B)
R1(A)
R1(B)
W1(A)
```

RUBRICS:

2a.

+2 For mentioning throughput/processing rate/any other synonym

+1 for any other partially correct reason (ANY reason related to efficiency, less delay etc, is fine)

0 for incorrect reason

2b.

+2 just for mentioning the schedule is serializable

+2 for explaining either swapping/sliding method

+1 for partial/incomplete explanations

0 for mentioning the schedule is non-serializable, irrespective of the explanation

Q3 [2+4 = 6 points].

What is the advantage of managing distributed transactions using the 2PC protocol?

A. A distributed transaction has a much smaller chance of failing, one account of one or more nodes in the cluster not being able to commit their transactions - this is because the coordinator polls all the nodes, and asks them to commit only if all are able to.

What are a couple of ways by which (even) 2PC can fail?

A. The coordinator node can fail. Or, network partitioning can occur during phase 2, when all the nodes are committing their transactions (meaning, a commit could fail after commencing).

Rubrics:

1. +2: Mentioning correct advantage of using 2PC protocol - close to the given solution. If solution is not clear and requires more explanation, deduct - 1.

2. +4: Minimum 2 points needed - blocking problem, network partitioning during phase 2, etc.

   If two similar points have been written using interchangeable words, consider it as one point only (but consider giving 3 points, instead of 2, or instead of 4)

Q4 [2 + 2*2 = 6 points].

What problem is the 2PL scheme designed to prevent?

A. The problem of lock starvation, ie. deadlock - on account of two or more in-progress transactions need to wait for each others' currently held locks to get released ('circular wait').

In the version of 2PL we looked at (called Conservative 2PL), there is a lock acquisition phase, operation phase, and a lock release phase (where the locks are being released before the transaction has been committed). With Conservative 2PL, what problem can occur during lock acquisition, and during lock release?

A. During lock acquisition, deadlocks can occur (but can be resolved, by rolling back o ending transactions). During release, if the uncommitted operation needs to be aborted, this will result in cell values possibly being reverted, which in turn could result in other transactions (which were granted locks to these cells a er our transaction began releasing them) having read incorrect (pre-rollback) data, which means THEY too need to be aborted (ie. it's the problem of 'cascaded aborts').

Rubric:

1. 2PL Scheme designed to prevent:
    a. Any answer explaining deadlock/ lock starvation/circular wait fetches 2 marks.
    b. If there is no near mention of deadlock -1 marks.
2. Conservative 2PL
    a. Lock Acquisition Phase
        i. Problem: Mention Deadlocks or any definition of the same. (2 marks)
    b. Lock Release Phase
        i. Problem: Explain process of cascading aborts. (2 marks)
        ii. If uncommitted data needs to be aborted, it may lead to other transactions getting aborted.(Another way of saying the same!(2 marks)
3. In 2nd part, If only one of the two processes [of lock acquisition and release phase] is explained (2/4 marks)

Q5 [1.5*4 = 6 points].

SQL is a set-oriented/declarative language, designed to operate on data held in tables. However it is not a complete programming language (such as C++, C#, JavaScript, Python, etc). What four (or more) features would you add to SQL, to make it be a more powerful/complete language?

A. Variables, flow control (branching, looping!), higher level types such as list and dictionary, import/include/source facility for reading in other scripts, static functions that operate on all rows of a table, functional constructs such as map() and filter(), etc…

Rubrics:

+ 1.5 for each of the above mentioned features.

If there are 4 valid features in the answer then the student gets 6 points.

Q6 [1.5*2 + 3 = 6 points].

A university chooses to represent courses and pre-reqs this way (CourseNumber is the PK):

```
[CourseNumber, Prereq1, Prereq2,
additional course-related columns...]
```

What are a couple of problems with representing the data like shown above?

A. One problem is wasted space - not all courses will have two prereqs; the other problem is the opposite - if a course has >2 prereqs, there is no way to list them.

What is a better design?

A. A separate 'prereqs' table, with just a pair of columns: [CourseNumber, Prereq]. Each prereq for a course, whether it is 1,2 or >2 for a given course, will get its own row. We could enhance the functionality with extra columns, eg. 'Mandatory' [a Boolean - if a prereq is not mandatory, it could be waived by a student's advisor or the course instructor].

Rubrics:

Problems: +1.5 for one reason (wasted space/ more than 2 prereq)

+1.5 for the other reason

Better design:

+1 for one suggestion for better design each upto a total of 3 (Separate 'prereq' table, Separate row for each prereq, extra column about mandatory or not) .

Any valid reason close to the one in the rubrics can also get +1.

Q7 [6 points].

The process of data modeling is one of abstraction, where, as a first step, relevant data-describing features ('columns', in a relational table) for an application need to be identified. Describe this process. Specifically, how does feature selection occur (where do features "live")?

A. Features are part of entities - and entities are gathered from business rules, policies, operating manuals, interviewing stakeholders (employees,

managers, other users of the DB), etc. Once entities are identified, "relevant" features for those entities need to be listed - these again come from business rules etc. In general, **features "live" in the minds of the users and designers** - they are NOT inherent to the world - we create features out of necessity - they help model whatever function/process/application we want to describe.

Rubrics:

No description of how to identify features. -2 marks

No mention of “entities” or similar. -1 mark

No mention of where features live. -2 marks

No mention of “minds of users and designers, or business rules” or similar. -1 mark