# COMP IT 420 Spring 2021 Final Examination

Started: May 19 at 8:16am

# Quiz Instructions

This in the final examination for the COMP IT 420 course. Please remember that you ***are not allowed to turn it in late***.

If you have requested DASS accommodations, you have an extra 24 hours to complete it.

You may submit the exam multiple times if you wish to save your place or make sure that major parts of it are completed before the deadline.

## Introduction

Many of the questions on this exam are based on two example databases, the Unified Department Management ERD schema and a subset of the "companies" MongoDB collection from Homework 5 (extra credit). Both the ERD and the document collection descriptions are included in this module as a separate descriptive PDF document and .drawio file.

***The ERD is incomplete. You will be filling it in to answer some questions on the exam and according to the business rules provided in the following Part 1 and the on the Review sheet provided on Canvas.***

---

## Part 1: ERD Analysis and Implementation

Please reference the University Department Management ERD and complete it based on the business rules below.

Your answer to Part 1 will be the submission of a completed ERD as an image. Please use .drawio to complete the ERD so that it is legible.

### Question 1 — 5 pts

Datatypes

Please finish the implementation of the missing datatypes in the diagram based on the following information:

1. Course_id consists of at most a four character department abbreviation and a 3 digit course number (i.e. "COMP420")
2. Sec_id is always a three digit number and will never be used in calculations
3. Class_building is a three-character designation (i.e. "SOL" = "Solano Hall")
4. Class_room includes the building designation and the highest possible room number is 9999.
5. Dept_budget has a maximum of $99,999,999.99 and instructor_salary has a maximum of $999,999.99.

◉ I have addressed the above datatype issues.

### Question 2 — 4 pts

Finish the implementation of the following associative entity:

ADVISES associates INSTRUCTOR and STUDENT. An INSTRUCTOR can have multiple advisees, each student can have only one advisor.

◉ I have finished the implementation of the ADVISES entity.

## Question 3 — 4 pts

Finish the implementation of the following associative entity:

TEACHES associates an INSTRUCTOR with a COURSE and a SECTION. An INSTRUCTOR can teach multiple sections and courses, a SECTION can only have one INSTRUCTOR, different INSTRUCTORS can teach different SECTIONS of the same COURSE.

◉ I have finished the implementation of the TEACHES entity.

## Question 4 — 15 pts

Some DBA took an early vacation (*cough* Greg) and forgot to finish the foreign key relationships.

Please add “FK” next to all foreign key references and make sure that all proper connectivity relationships (Crow’s Foot) are present.

Also, make sure that weak and strong relationship lines are correct (dotted vs. solid).

***(Hint: each table prefaces its attributes, therefore, different prefixes [‘student’, ‘dept’, etc.] probably indicate information from other tables in some cases. There is a unary relationship [an entity connects to itself] in at least one table.)***

◉ I have finished Greg's job for him and identified all the proper relationships.

## Question 5 — 0 pts

Please submit your completed ERD here.

Upload COMP_420_FINAL_ERD.png ×

*Your file has been successfully uploaded.*

# Part 2: SQL

Please reference your completed University Department Management ERD to answer the following questions.

## Question 6 — 8 pts

Please complete the following SQL query so that it returns the name and total number of credits each instructor taught in the Fall 2020 and Spring 2021 semesters if they taught more than four credits.

Sort alphabetically by instructor name.

Note that each section counts as a full course credit, so if someone teaches two sections of COMP420 and COMP420 is four course_credits, then that is eight total credits.

SELECT instructor_name, ***(1)*** count ( ***(2)*** course_credits ) FROM course

JOIN teaches using( ***(3)*** course_id )

JOIN ***(4)*** section using(sec_id)

JOIN instructor using( ***(5)*** sec_id )

WHERE ( ***(6)*** sec_semester = " ***(7)*** Fall " AND sec_year = 2020) ***(8)*** AND

(sec_semester = "Spring" ***(9)*** AND sec_year = 2021)

***(10)*** GROUP BY instructor_id

***(11)*** HAVING sum(course_credits) > ***(12)*** 4

ORDER BY ***(13)*** instructor_id ***(14)*** ASC ;

## Question 7

8 pts

The function "validate_credits" takes a course and a number of credits and checks to see whether the credit assignment aligns with the number of hours each course section is offered per week. For simplicity, each hour of class time is equal to a valid credit hour.

Please fill in the blanks in the validate_credits function below to ensure that courses have correct credits assigned.

TIMESTAMPDIFF takes an interval ("MINUTE", "SECOND", etc.), a start time and an end time and returns the number of units for that interval as an INT.

If a FLOAT value is equal to an INT value, e.g. 1.0 = 1, then MySQL will evaluate to true even though the types are different.

DELIMITER ***(1)*** //

CREATE FUNCTION validate_credits (***(2)*** course_to_check VARCHAR(7), credits ***(3)*** tinyint )

RETURNS bool

```
BEGIN
    -- declare variables to hold section total minutes per week
DECLARE sec_tot_week_minutes INT;
-- populate variable with section minutes per week
SELECT (4) [sum] (sec_time_length) INTO sec_tot_week_minutes
FROM
(
SELECT TIMESTAMPDIFF(MINUTE, time_slot_start, time_slot_end) AS (5) [sec_time]
FROM section
JOIN (6) [time_slot] USING(time_slot_id)
WHERE (7) [course_id] = course_to_check
);
    -- compare the input credit amount with the credit hours
    -- computed from the section lengths in minutes
    RETURN (credits = sec_tot_week_minutes / (8) [sec_time] )
END //
```

## Part 3: Transaction Management

Refer to the University Department Manager ERD for the following two questions.

### Question 8

**8 pts**

The transaction below enables a student (id 1234) enrollment in a specific course, "COMP520", section "001". In order to enroll the following conditions must be met:

1. A student must be affiliated with the department of the course.
2. A student's total credits cannot exceed 180.
3. The course enrollment cannot exceed the capacity for the classroom.
4. The total credits for the current semester (Spring 2020) for a student cannot exceed 18.
5. A student must have taken the required prerequisite for COMP520.

(Note: If a condition has failed, a ROLLBACK call is expected.)

```
-- Declare temporary variables
```

```
DECLARE stu_tot_credits INT;
DECLARE stu_current_credits INT;
DECLARE stu_dept INT;
DECLARE comp_520_creds INT;
DECLARE comp_520_dept INT;
DECLARE class_capacity INT;
DECLARE current_class_capacity INT;
DECLARE (1) course_id VARCHAR(7);

-- Initiate the transaction
(2) START TRANSACT ;

-- Get the total number of current credits for the student
-- in the current semester
SELECT sum( (3) course_credits ) INTO stu_current_credits
FROM student JOIN enroll USING(student_id)
(4) JOIN course USINC
JOIN section USING(sec_id)
WHERE (5) sec_semseter = 'Sp ;

-- Check that current credits do not exceed condition D
IF (6) stu_current_credits > 18 THEN
    ROLLBACK;
END IF;

-- Get the total credits for student
SELECT student_tot_creds INTO (7) stu_tot_credits FROM student WHERE student_id = '1234';
-- Get the total course_credits for COMP520
SELECT course_credits INTO comp_520_creds FROM course WHERE course_id = 'COMP520';

-- Make sure that total credits do not exceed condition B
IF (8) stu_tot_credits + comp_520_creds > 180 THEN
```

ROLLBACK;

END IF;

-- Get the dept_ids to compare for condition A

SELECT dept_id INTO stu_dept FROM student WHERE student_id = '1234';

SELECT dept_id INTO comp_520_dept FROM course WHERE course_id = 'COMP520';

-- Compare dept ids

IF stu_dept != comp_520_dept THEN

***(9)*** ROLLBACK ;

END IF;

-- Get class room capacity for section

SELECT class_capacity FROM classroom

JOIN section ON ***(10)*** sec_room AND classroom.class_building = section.sec_building WHERE ***(11)*** sec_id = '001' AND ;

SELECT count(*) INTO ***(12)*** current_class_capa FROM enroll WHERE course_id = 'COMP520' AND sec_id = '001';

-- Compare capacities to check condition C

IF current_class_capacity + 1 > ***(13)*** 180 THEN

ROLLBACK;

END IF;

-- Get course prerequisite

SELECT course_prereq_id INTO prereq FROM course WHERE course_id = 'COMP520';

-- Check for condition E

IF prereq IN (SELECT course_id FROM enroll JOIN student USING(student_id) WHERE student_id = ***(14)*** 1234 ) THEN

INSERT INTO enroll VALUES ('1234', ***(15)*** COMP520 , '001');

***(16)*** ELSE

ROLLBACK;

END IF;

***(17)*** COMMIT ;

## Question 9

**5 pts**

| **TRANSACTION 1** | **TRANSACTION 2** |
|---|---|
| START TRANSACTION | START TRANSACTION |
| validate_credits("COMP420", 4) | |
| UPDATE course SET course_credits = 4 WHERE course_id = "COMP420"; | DECLARE comp_420_cred; |
| | SELECT course_credits INTO comp_420_cred FROM course WHERE course_id = “COMP420”; |
| | UPDATE student SET student_tot_creds += comp_420_cred WHERE student_id = 1234; |
| ROLLBACK | |
| | COMMIT |

Based on the transactions above, which concurrency error is present (read on uncommitted data, lost update, or inconsistent retrieval)? Read on uncommitted data

What is the minimum isolation level (read uncommitted, read committed, repeatable read or serializable) needed to avoid this problem?

Read Uncommitted

## Question 10

**3 pts**

Please explain your choices for the concurrency error and isolation level in the previous question. Make sure to reference which transaction is doing what actions, and how the error occurred.

There is a read on Uncommitted data or a dirty read because t1 updated the data that t2 was trying to access. The transaction with the uncommitted data was then completed with a rollback, this made it so that t2 had some data which never existed.

The minimum isolation level is read uncommitted because that is the lowest isolation level that Read on uncommitted data are allowed.

p 67 words </>

## Part 4: Optimization

Refer to the University Department Management ERD in addition to the following query to answer the questions in this part.

There are 300 courses at the university with a combined total of 650 sections running during any given semester. The course with the most sections has at most, 4 sections per semester. The university has existed for 6 years, with two semesters each year, and no instructor has ever taught more than 3 sections a semester. There are 100 instructors in the entire database. Assume there is indexing on course_id in SECTION and instructor_id in INSTRUCTOR.

```
1. SELECT course_id, sec_id FROM section s
2. JOIN teaches t ON s.course_id = t.course_id AND s.sec_id = t.sec_id
3. JOIN instructor USING(t.instructor_id)
4. WHERE instructor_id = 75
5. AND course_id = “COMP420”;
```

### Question 11 2 pts

For the query above, which part of the query should be executed first assuming each line will run independently?

- ○ 1
- ◉ 2
- ○ 3
- ○ 4
- ○ 5

## Question 12

2 pts

What is the maximum number of I/O operations your chosen line will contribute to the query execution?

Total I/O = index reads + table reads if an index is used, otherwise Total I/O = total table reads. Also remember to include the table reads your line will contribute to any JOINs.

101

## Question 13

4 pts

If you were adding a new index on course_name which data structure (B-tree, hash table or bitmap index) would make the most sense and why?

(Please make sure to refer to data sparsity in your answer.)

If I were adding a new index on course_name I would choose B-Tree data structure because B-tree is mostly used for indexing. B-tree provides sequential search capabilities in addition to the binary search, which give the database more control to search non-index values in a database. Its not like a Hast indexing structure, as it stores data in a ordered way which makes it faster for row retrieval when the selection conditions include things line inequalities and prefixes.

p 78 words </>

## Part 5: NoSQL

Refer to the Companies JSON description to answer the following questions.

### Question 14 — 3 pts

Write a MongoDB query to find all the companies founded after the year 2005 that currently have more than 10000 employees.

Edit View Insert Format Tools Table

12pt Paragraph

```
db.companies.find(
{founded_year: {$gt: 2005} }
{num_employees: {$gt: 10000} }
)
```

p 9 words

### Question 15 — 5 pts

1. Fill in the following aggregate pipeline query to find the total number of employees working at companies that start with the letter “F”.

```
db.companies.aggregate([
    {$(1) match : {(2) name : {$(3) search : /^F/}}},
    {$group: {
        _id: “total_employees”,
        total:{$(4) sum : “(5) num_employees ”}}}
])
```

## Question 16

4 pts

1. Fill in the following MapReduce to find the same information (total number of competitors for each company).

```
db.companies.mapReduce((1) mapComp , reduceComp,
 {output: “total_comp_per_company”})

function mapComp(){
    var i;
    for( i = 0; i < this.(2) competitions .length; i++){
        emit(this.competitions[ i ].(3) length , 1);
}
}

function reduceComp(key, (4) values ){
    return Array.sum(values);
}
```

## Question 17

5 pts

# Part 6: Concurrency

| *t* | T1 | T2 | T3 | T4 |
|---|---|---|---|---|
| 1 | | Write(C) | Read(A) | Write(B) |
| 2 | Read(C) | | | Read(C) |
| 3 | Write(A) | | Write(D) | |
| 4 | | Write(B) | | |

Based on the above transaction schedule, please construct a precedence graph.

***Please upload an image of your precedence graph.***

Upload 17.png ×

*Your file has been successfully uploaded.*

## Question 18

1 pts

The above transaction schedule is conflict-serializable.

- ○ True
- ◉ False

## Question 19

2 pts

Please explain your answer to the previous question regarding conflict serializability for the above schedule.

Edit View Insert Format Tools Table

12pt Paragraph

There is a cycle in the precedence graph between T2 and T4 so it is not conflict serializable.

p 18 words

## Question 20

2 pts

If two-phase locking (2PL) is implemented, will there be a deadlock on the above schedule?

- ◉ Yes
- ○ No

## Question 21

2 pts

Please explain your previous answer regarding a potential deadlock.

Edit View Insert Format Tools Table

12pt Paragraph

Yes, because if T1 requires the lock for C at t = 2, then T4 cannot complete at t = 2, because it is in a deadlock. C has already been logged to T2, so until a rollback there is a situation of deadlocking.

p 44 words

## Question 22

8 pts

Transaction Schedule

| *t-index* | T1 | T2 | T3 |
|---|---|---|---|
| 1 | Write(A) | Read(B) | |
| 2 | | | Write(B) |
| 3 | | | Read(A) |
| 4 | Write(C) | Read(C) | |
| 5 | Read(C) | | |
| 6 | | Write(B) | |

Transaction Status Table

| *t-index* | Transaction | Object | R-TS | W-TS |
|---|---|---|---|---|
| 1 | 1 | A | 0 | 1 |
| 1 | 2 | B | 2 | 0 |
| | | | | |

| | | | | |
|---|---|---|---|---|
| 2 | 3 | B | 0 | 3 |
| 3 | 3 | X | X | X |
| 4 | 1 | C | 0 | 1 |
| 4 | 2 | X | X | X |
| 5 | 1 | X | X | X |
| 6 | 2 | X | X | X |

According to the transaction schedule above, please fill in the table above by following a simple timestamp ordering. If a specific row of the transaction schedule results in a rollback, please put "X" in both the R-TS and W-TS cells for that row in the corresponding table.

The rules for simple timestamp ordering are as follows:

**R-TS** (Read Timestamp)

object tagged with transaction timestamp of last read

**W-TS** (Write Timestamp)

object tagged with transaction timestamp of last write

The timestamp of each transaction above is equal to its transaction number for convenience.

If a transaction wants to read an object A and:

TS(Ti) < W-TS(A)

reject read and rollback Ti

TS(Ti) >= W-TS(A)

read A and set R-TS(A) to equal max(R-TS(A), TS(Ti))

If a transaction wants to write to object A and:

TS(Ti) < R-TS(A)

reject write and rollback Ti

TS(Ti) < W-TS(A)

reject write and rollback Ti

Otherwise, execute write and set W-TS = TS(Ti).

**Question 23** 0 pts

If you are for some reason unable to fill in the right column for the above question, please put your answers in this column below

| W-TS |
| --- |
| [ Select ] |
| [ Select ] |
| [ Select ] |
| [ Select ] |
| [ Select ] |
| [ Select ] |
| [ Select ] |
| [ Select ] |

## Extra Credit: Normalization

Reference the University Department Management ERD for the following questions:

**Question 24** 0 pts

(6pts) There are potential entities in the diagram that would benefit from the use of surrogate keys. Propose 2 potential surrogate keys and explain how they might help with system retrieval and avoid data anomalies.

Edit View Insert Format Tools Table

12pt Paragraph

I would propose a surrogate key for the sec_id. If the school logic changes at some point, and new sections need to be added or deleted then a surrogate key would help keep the information of previous courses with the same section ID. Also if the sec_id changed there is a lot of foreign key references to the value and you would have to update a natural key across all foreign key relationships.

Another surrogate key could be course_id with the same premise as above.

p 85 words

## Question 25

0 pts

(6pts) Identify 2 instances of denormalization in the ERD and propose a normalization solution for each. Please specific how it could improve specific aspects of the overall system design AND why the denormalization was likely used in the first place.

Edit View Insert Format Tools Table

12pt Paragraph

There is a instance of denormalization in the ENROLL table. If we wanted to access student ID corresponding to a specific grade, we can do that. They introduced enroll grade column to store information about the grade enrolled.

p 38 words

Quiz saved at 9:28pm Submit Quiz