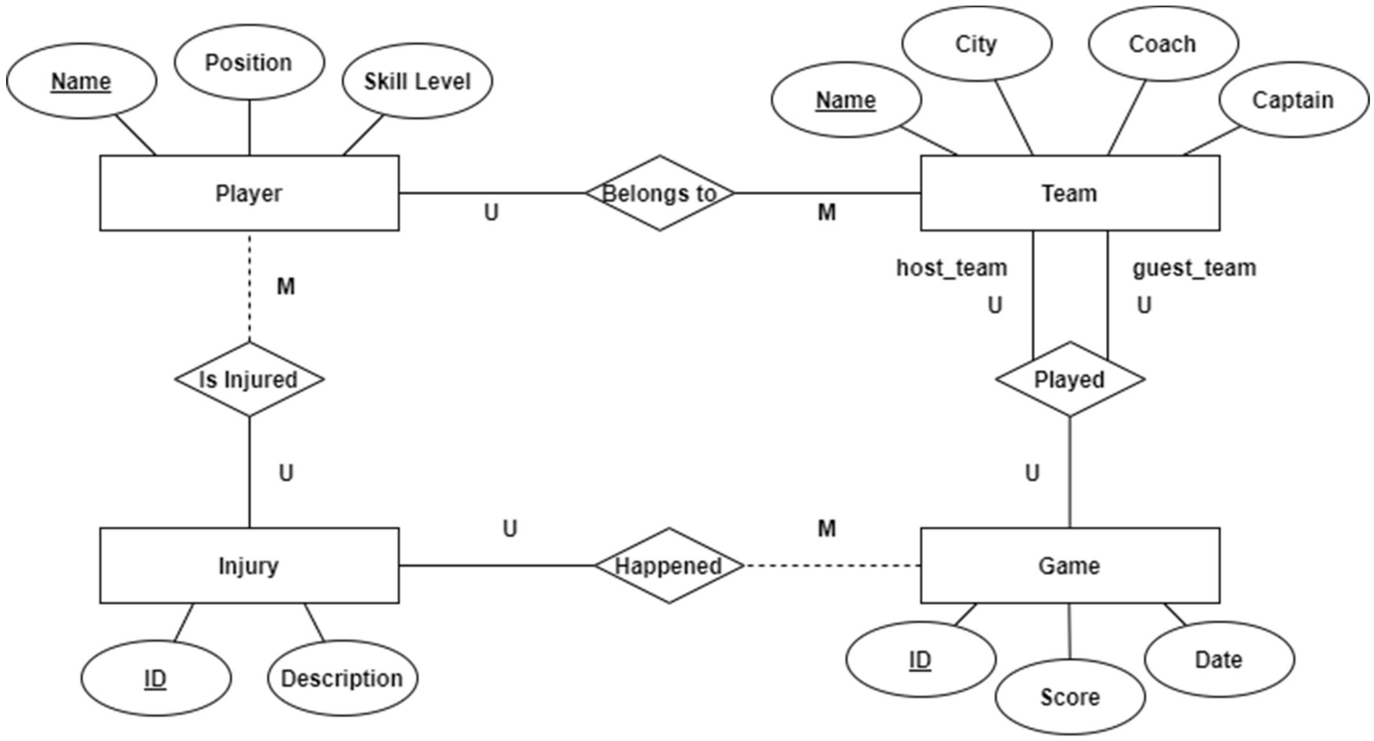
NAME: Sean Gordon (Sgordon4)

COM S 363 Final

Problem	Max Points	Points
1	8	
2	7	
3	18	
4	15	
5	15	
6	8	
7	9	
8	20	
Total	100	

1. (8 points) Construct an ER diagram for a simple database that records the following information for the United Soccer League (USL).
   a) There are many teams in the league.
   b) Each team has a name, a city, a coach, a captain, and a set of players
   c) Each player belongs to only one team.
   d) Each player has a name, a position (such as left wing striker or goalie), a skill level.
   e) A player may be injured. Each injury has an ID, a description, and the game when it happens.
   f) A team captain is also a player.
   g) A game is played between two teams (referred to as host_team and guest_team) and has a date (such as May 4, 2020) and a score (such as 4 to 2)



2. (7 points) Consider a relation R with attributes A, B, C, D, E, and G, and the set of dependencies F = $\{AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A, E \rightarrow G\}$. Answer the following questions.

   a) (3 points) Find all keys that contain only two attributes

      {AB}+ = {ABCDEG}
      {AC}+ = {ABCDEG}
      {AD}+ = {ADEG}
      {AE}+ = {AEG}
      {AG}+ = {AG}
      {BC}+ = {ABCDEG}
      {BD}+ = {BD}
      {BE}+ = {BEG}
      {BG}+ = {BG}
      {CD}+ = {CD}
      {CE}+ = {CEG}
      {CG}+ = {CG}
      {DE}+ = {DEG}
      {DG}+ = {DG}
      {EG}+ = {EG}

b) (4 points) Suppose R is decomposed into R1(ABC), R2(ACDE), and R3(ADG). Is this a lossless join decomposition? Explain your answer in details.

This decomposition is indeed lossless, as all conditions are satisfied –

1) $R1 \cup R2 \cup R3 = R$ ✓

2) $R1 \cap R2 \neq \emptyset$, $R2 \cap R3 \neq \emptyset$ ✓

3) All sub relations are linked by attribute A, so there are no disconnects. ✓

3. (18 points) query processing. We have the following relations.

- **students**

snum	name	gender
1001	Randy	M
1005	Nicole	F

- **departments**

code	name	college
401	Computer Science	LAS
402	Mathematics	LAS
403	Chemical Engineering	Engineering
404	Landscape Architect	Design

- **degrees**

name	level	department_code
Computer Science	BS	401
Software Engineering	BS	401
Computer Science	MS	401
Computer Science	PhD	401
Applied Mathematics	MS	402
Chemical Engineering	BS	403
Landscape Architect	BS	404

- **major**

snum	name	level
1001	Computer Science	BS
1005	Applied Mathematics	MS

- **minor**

snum	name	level
1005	Computer Science	MS
1001	Software Engineering	BS

- **courses**

number	name	description	credithours	level	department_code
113	Spreadsheet	Microsoft Excel and Access	3	Undergraduate	401
311	Algorithm	Design and Analysis	3	Undergraduate	401

531	Theory of Computation	Theorem and Probability	3	Graduate	401
363	Database	Design Principle	3	Undergraduate	401
412	Water Management	Water Management	3	Undergraduate	404
228	Special Topics	Interesting Topics about CE	3	Undergraduate	403
114	Calculus	Limit and Derivative	4	Undergraduate	402

- **register**

snum	**course_number**	**regtime**	**grade**
1001	363	Fall2015	3
1005	412	Spring2015	4

1) (10 points) Please show the results for the following relational algebra expressions

a. $\pi_{level}(degrees)$

b. $\sigma_{number<300}(courses)$

c. $department \times major$

d. $department \bowtie major$

e. $degree \bowtie_{degree.name=majo\ \ .name} major$

(a) $\pi_{level}(degrees)$

level
BS
MS
PhD

(b) $\sigma_{number<30}\ (courses)$

number	**name**	**description**	**credithours**	**level**	**department_code**
113	Spreadsheet	Microsoft Excel and Access	3	Undergraduate	401
228	Special Topics	Interesting Topics about CE	3	Undergraduate	403
114	Calculus	Limit and Derivative	4	Undergraduate	402

(c) $department \times major$

code	d.name	college	snum	m.name	level
401	Computer Science	LAS	1001	Computer Science	BS
401	Computer Science	LAS	1005	Applied Mathematics	MS
402	Mathematics	LAS	1001	Computer Science	BS
402	Mathematics	LAS	1005	Applied Mathematics	MS
403	Chemical Engineering	Engineering	1001	Computer Science	BS
403	Chemical Engineering	Engineering	1005	Applied Mathematics	MS
404	Landscape Architect	Design	1001	Computer Science	BS
404	Landscape Architect	Design	1005	Applied Mathematics	MS

(d) $department \bowtie major$

code	d.name	college	snum	level
401	Computer Science	LAS	1001	BS

(e) $degree \bowtie_{degree.name=major.name} major$

name	level	department_code	snum
Computer Science	BS	401	1001
Applied Mathematics	MS	402	1005

2) (8 points) Please write the relational algebra expressions and sql code for the following queries.

a. The course numbers and names of all courses offered by Randy's home department
b. The college(s) that have student registered.

```
(a) π number, course_name (
        σ student_name=='Randy' (
            (ρ name → student_name(students)) ⋈
            (major) ⋈ (degrees) ⋈
            (ρ name → course_name, level → course_level(courses))
    ))

    SELECT DISTINCT
        courses.number as CourseNum,
        courses.name as CourseName
    FROM courses
        INNER JOIN degrees ON (
            courses.department_code=degrees.department_code
        )
        INNER JOIN major ON (
            degrees.name=major.name
        )
         INNER JOIN students ON (
            major.snum=students.snum
        )
    WHERE
        students.name='Randy';
```

(b) π college (
```
            (ρ name → student_name(students))⋈
            (register)⋈
            (ρ name → course_name, number → course_number,
                        department_code → code(courses))⋈
            (departments)
    )

    SELECT DISTINCT
            departments.college as College
    FROM departments
            INNER JOIN courses ON (
                        departments.code=courses.department_code
            )
            INNER JOIN register ON (
                        courses.number=register.course_number
            )
            INNER JOIN students ON (
                        register.snum=students.snum
            );
```

4. (15 points) For each of the following schedules, determine what schedule(s) it is. Please explain briefly

S1

T1	T2
R(A)	
	W(B)
W(A)	
commit	
	commit

a) S1 is a serial schedule. Yes[ ] No[X]
b) S1 is a strict schedule. Yes[X] No[ ]
c) S1 is a serializable schedule. Yes[X] No[ ]

a) The actions are interleaved; thus, this cannot be serial.
b) Nowhere in this schedule does one transaction read a value, and the other writes to that value before the first commits/aborts.
c) This schedule is equivalent to a serial version.

S2

T1	T2
R(A)	
	R(A)
W(A)	
	W(B)
commit	
	commit

d) S2 is a serial schedule. Yes[ ] No[X]
e) S2 is a strict schedule. Yes[ ] No[X]
f) S2 is a serializable schedule. Yes[X] No[ ]

a) The actions are interleaved; thus, this cannot be serial.
b) T2 reads A, then T1 writes A before T2 commits/aborts.
c) This schedule is equivalent to a serial version.

S3

T1	T2	T3
R(A)		
R(B)		
	R (B)	
	commit	
		R(A)
W(A)		
commit		
		commit

g) S3 is a serial schedule. Yes[ ] No[X]
h) S3 is a strict schedule. Yes[ ] No[X]
i) S3 is a serializable schedule. Yes[X] No[ ]

a) The actions are interleaved; thus, this cannot be serial.
b) T3 reads A, then T1 writes A before T3 commits/aborts.
c) This schedule is equivalent to a serial version.

5. (15 points) Strict 2PL uses a lock table to track 1) which data object is locked; 2) which transaction owns which lock; 3) which transaction is waiting for a lock on which object. Consider the following two transactions. For each step of their execution, write down the status of the corresponding lock table. An execution can result in deadlock. When this happens, you write own "deadlock" and ignore the rest steps.

S1

T1	T2
X(A)	
R(A)	
	S(A)
W(A)	
commit	
	R(A)
	commit

1) T1.X(A)

Data	Lock	Owner	Waiting
A	X	T1	

2) T1.R(A)

Data	Lock	Owner	Waiting
A	X	T1	

3) T2.S(A)

Data	Lock	Owner	Waiting
A	X	T1	T2

4) T1.W(A)

Data	Lock	Owner	Waiting
A	X	T1	T2

5) T1.commit

Data	Lock	Owner	Waiting
A	S	T2	

6) T2.R(A)

Data	Lock	Owner	Waiting
A	S	T2	

7) T1.commit

Data	Lock	Owner	Waiting

6. (8 points) Draw the Precedence Graph for the following schedules. Are they conflict serializable?

S1

T1	T2	T3
R(A)		
R(B)		
	R (B)	
	commit	
		R(A)
		commit
W(A)		
commit		

$$\begin{tikzcd}
T1 & & T2 \\
& T3 \arrow[ul]
\end{tikzcd}$$

The graph is **acyclic**; thus, the schedule **is** conflict serializable

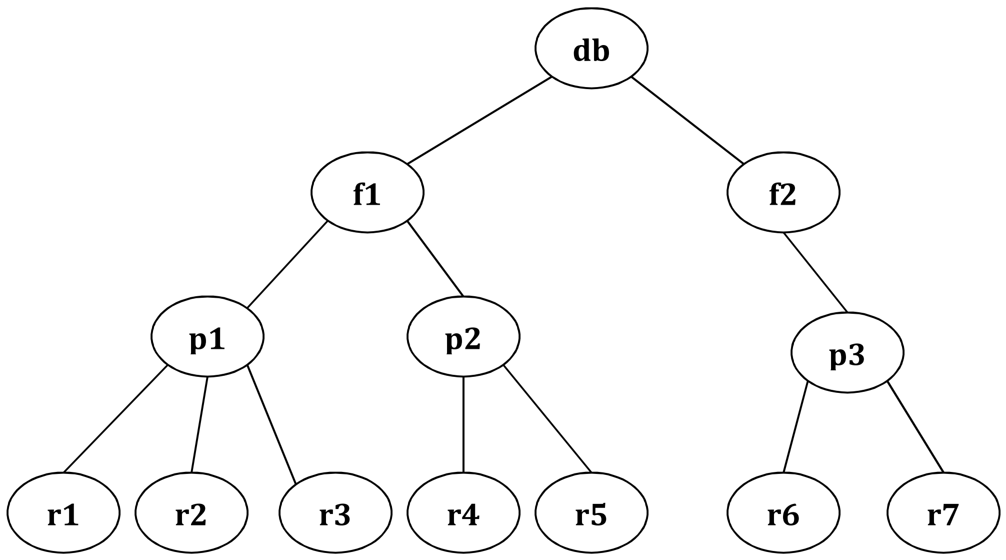
S2

T1	T2	T3
R(A)		
	R (B)	
	W(B)	
R(B)		
		R(A)
		W(A)
W(A)		
		commit
	commit	
commit		

$$\begin{tikzcd}
T1 \arrow[dr, bend right] & & T2 \arrow[ll] \\
& T3 \arrow[ul, bend right]
\end{tikzcd}$$

The graph is **cyclic**; thus, the schedule **is NOT** conflict serializable

7. (9 points) For better current execution, multiple-granularity locking (MGL) introduces three new types of locks, IS, IX, and SIX. Consider the following tree of objects, where each node contains all its children.



a) (3 points) Transaction T1 needs to read object r1. What lock(s) on which node(s) does it need to acquire?
b) (3 points) Transaction T2 needs to write object r4 and r5. What lock(s) on which node(s) does it need to acquire?
c) (3 points) Transaction T3 needs to read r6 and r7. What lock(s) should it acquire on object p3?

(a) IS lock on db, f1, and p1.
S lock on r1.

(b) IX lock on db, f1, and p2.
X lock on r4 and r5.

(c) IS lock.

8. (20 points) Database for COVID-19

   We are in need of a database for COVID-19 reports that can answer the following questions
   For each day, in each county in IA

   1. How many tests are conducted?
   2. How many additional confirmed cases?
   3. How many additional hospitalized patients?
   4. How many additional death?
   5. How many additional recovered patients?
   6. How many additional discharged patients?

We also need to report the demographic information of those patients including age group, gender, ethnicity, and race. As outbreaks occurred in long-term care facilities, it's better to know which patients are related with LTC (staff/resident) as well.

This database will have three parties of users: test labs, hospitals, and the government. The test labs will handle all COVID-19 tests. They will enter the test results for every test and the basic demographic information of the testees. A person can get multiple tests. If a person is tested positive, then it is a confirmed case. If a confirmed patient is later tested negative, then this patient is recovered. The hospitals will update their patients' status (hospitalized/discharged/death). Hospitals also need to query the test labs for the patients' test results. The government will query the database to update the daily reports: for example https://coronavirus.iowa.gov/.

Q1 (4 pts). Please design a database and explain your design (also any additional assumptions). You can get extra credits if your design is more comprehensive than the basic needs described above. Your design should not require test labs or hospitals to provide summarized numbers everyday so that they can focus on their jobs.

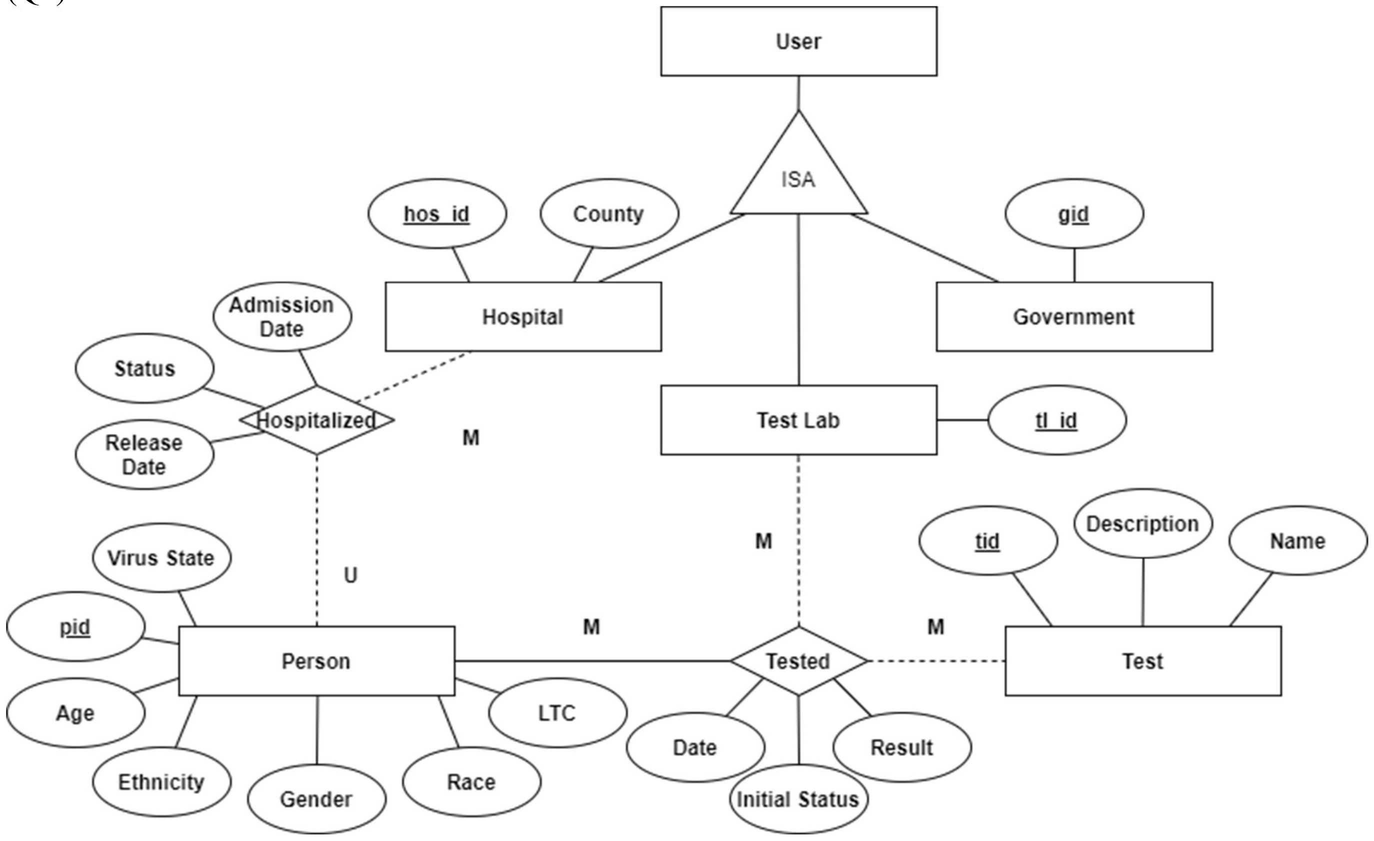
Q2 (4 pts). Please provide examples SQL codes to the test labs and hospitals of how to insert and update records in your designed database.

Q3 (4 pts). Please provide SQL codes for finding the county in Iowa which has the highest number of confirmed patients who are elderly and LTC residents based on your database.

Q4 (4 pts). The hospitals will submit many queries for the test results of their patients, and test labs will update the test results very frequently. Please explain what you plan to choose for file format and indexing to improve the query responding time?

Q5 (4 pts). Different users may access your database concurrently. Please explain how will you handle concurrency?

(Q1)



Person: Stores information about an individual person. 'Age' is represented as a number, with ages 65+ considered elderly. LTC represents whether the individual is a member of an LTC, taking the values of no/staff/resident. A person's 'Virus State' refers to whether the individual is infected, with values negative/positive. 'Virus State' defaults to negative.

Test: Stores information about an individual type of test. Multiple tests have been created to detect the virus, so a test's name and any description are recorded.

User: A user of the database.

Hospital: A type of user, a hospital stores the county it is located in.

Test Lab: A type of user.

Government: A type of user.

Tested: Stores information about an instance of a test being administered to a patient in a test lab. The date the testing occurs is recorded, as well as the initial state of the patient (negative/positive) and the test results (negative/positive).

Hospitalized: When a person is admitted to a hospital for the virus, the date of admission is recorded, and 'Status' is set to 'hospitalized'. When the patient is released, either by discharge or death, the release date is recorded, and 'Status' is changed to either 'discharged' or 'death'.

(Q2)

```
INSERT INTO person (pid, age, ethnicity, gender, race, ltc, virus_state) VALUES
(1, 66, ‘Hispanic’, ‘F’, ‘Caucasian’, ‘no’, ‘negative’);

INSERT INTO test (tid, name, description) VALUES (1, ‘TheGoodTest’, ‘This test is good’);

INSERT INTO hospital (hos_id, county) VALUES (1, ‘Winnebago’);


INSERT INTO test_lab (tid) VALUES (1);


INSERT INTO government (gid) VALUES (1);


INSERT INTO tested (date, initial_status, result) VALUES (‘2020-06-10’, ‘positive’, ‘positive’);


INSERT INTO hospitalized (admission_date, status, release_date) VALUES
(‘2020-06-12’, ‘hospitalized’, NULL);
```

(Q3)

```
SELECT
        hospital.county
FROM hospital
        INNER JOIN hospitalized ON (
        hospital.hos_id = hospitalized.hos_id
        )
        INNER JOIN person ON (
        hospitalized.pid = person.pid
        )
WHERE
        person.age >= 65 AND
        person.ltc = ‘resident’
GROUP BY hospital.county
ORDER BY count(hospital.county) DESC LIMIT 1;
```

(Q4)

To combat the workload put on the database, this application will use the well supported .csv file format, with ‘tested’ indexed on date, ‘hospitalized’ indexed on admission_date, and ‘person’ indexed on virus_state. These attributes are the most likely to be queried.

(Q5)

Concurrency control will be enforced using strict 2pl for the best all-around performance. This will ensure user transactions do not overlap and cause issues and allows for some error.