Question 1. SQL (24 points, 8 each part) The Wrecks 'R Us Auto Repair Shoppe has a database to keep track of cars, repair orders, and repair technicians. The database has the following tables:

CAR(vin, make, model, year) -- car vehicle ID #, manufacturer, model, and year; for example: (17234, Toyota, Prius, 2008) or (38953, Chevy, Nova, 1968)
TECHNICIAN(techid, name, yearhired) -- technician ID number, name, and year hired
CANFIX(techid, make, model) -- table listing the car makes and models that a technician can fix
ORDER(num, vin, techid, year, month, day) -- repair order #, car ID, technician ID, and date

In these relations, vin is a foreign key in ORDER referring to the CAR relation, and techid is a foreign key in the CANFIX and ORDER relations referring to the TECHNICIAN relation. The keys in each relation are underlined.

(a) Write a SQL query that produces a list of the names of all technicians who repaired a Toyota car (i.e., make = 'Toyota') in the year 2004. The list should be sorted by technician name and should not contain duplicates.

```
SELECT DISTINCT t.name
FROM technician t, order r, car c
WHERE c.make = 'Toyota'  and c.vin = r.vin and r.year = 2004 and r.techid = t.techid
ORDER BY t.name
```

Several people used GROUP BY, which also works, but the ORDER BY clause is still needed and must follow GROUP BY.

```
SELECT t.name
FROM technician t, order r, car c
WHERE c.make = 'Toyota'  and c.vin = r.vin and r.year = 2004 and r.techid = t.techid
GROUP BY t.name
ORDER BY t.name
```

(continued next page)

Question 1. (cont) Schemas repeated for convenience:

CAR(vin, make, model, year) -- car vehicle ID #, manufacturer, model, and year
TECHNICIAN(techid, name, yearhired) -- technician ID number, name, and year hired
CANFIX(techid, make, model) -- table listing the car makes and models that a technician can fix
ORDER(num, vin, techid, year, month, day) -- repair order #, car ID, technician ID, and date

(b) Write a SQL query that reports the car make and model that accounts for the most repair orders in the database. If two or more car make/model pairs are tied for the maximum, your query can list any one of them or all of them, at your discretion.

```
SELECT make, model
FROM car c, order r
WHERE c.vin = r.vin
GROUP BY make, model
HAVING count (*) >= ALL (
        SELECT count(*)
        FROM car c2, order r2
        WHERE c2.vin = r2.vin
        GROUP BY make, model
)
```

Solutions that used SQL Server's top function or the limit clause in postgres or mysql also received credit.

(c) Write a SQL query that lists the names of all technicians who were hired before 2005 and can fix a Ford Mustang (i.e., make is 'Ford' and model is 'Mustang').

```
SELECT t.name
FROM canfix f, technician t
WHERE t.techid = f.techid and f.make = 'Ford' and f.model = 'Mustang'
        and t.yearhired < 2005
```

Question 2. Conceptual design (20 points) We would like to design a database to keep track of students who participate in intramural sports.

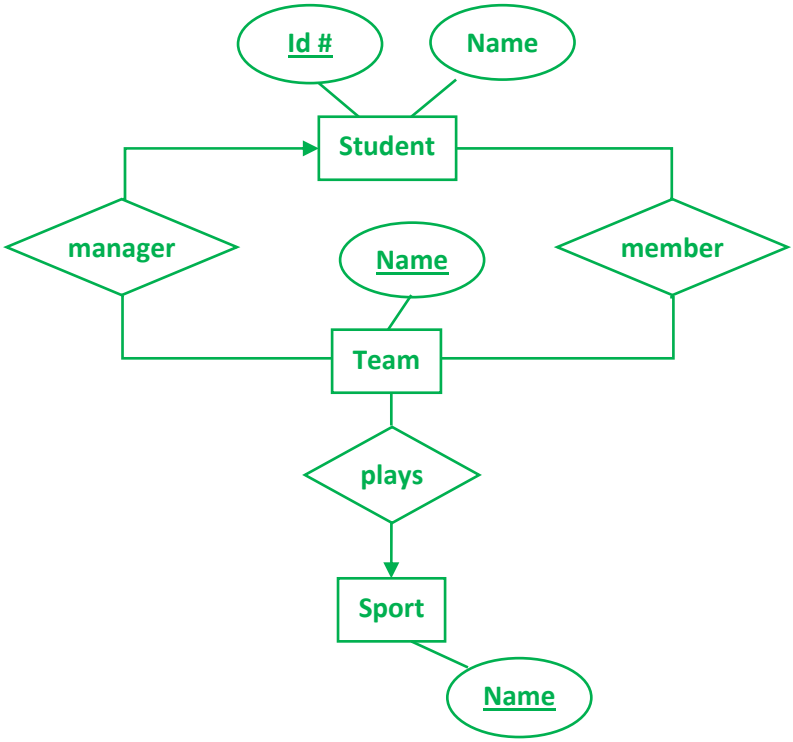
(a) Give an E/R diagram that captures the following entities and relationships:

Entities:

- Student(name, id#)
- Sport (name)
- Team(name, sport)

Relationships:

- Each Team has team members, who are Students, and a single manager, who is also a Student
- Each Sport has one or more Teams
- Each Team belongs to exactly one Sport
- A Student can be a member of any number of teams and can manage any number of teams, including teams that he/she is a member of.



Grading notes: Another way to diagram this would be to use is-a relations to define two subclasses of students: team members and managers. We gave full credit if this was done well, although most people had a simpler solution like the one above.

In a couple of places it would make sense to use a rounded arrow to indicate "exactly one" instead of a regular arrow to indicate many-to-one. We allowed either, since both captured the intended many-to-one semantics, which was the major point, and was adequate for the exam.

(continued next page)

Question 2 (cont.) (b) Give a relational schema that captures your E/R diagram from part (a). You should give the table and attribute names and clearly indicate which attributes are keys and foreign keys in the various tables. You do **not** need to give SQL CREATE TABLE statements for your tables.

As in the homework, R.x indicates that attribute x is a foreign key in relation R. Primary keys are underlined.

Student(id, name)
Team(name, Student.id, Sport.name) -- manager and plays relations folded into this table
Sport(name)
Member(Student.id, Team.name)

Some answers used both the team and sport names as keys for the Team relation, which would allow the same name to be used for teams in different sports. That received full credit if it was done properly. In a real situation we would go back to the client and ask whether team names were intended to be unique across all sports.

Question 3. BCNF (20 points) As they do every four years, this summer Seattle Opera is presenting Wagner's *Der Ring des Nibelungen*. This 4-opera mini-series has a huge cast, and one of the major donors to the company would like a database to keep track of the characters. One of the summer interns was given the job and produced a table named RING. It lists the characters and information about them, such as their voice part, what they have power over (fire, love, the magic sword, etc.), the name of their residence, and its address.

RING

Character	Voice	Power	Residence	Address
Wotan	Baritone	Light	Valhalla	Rainbow Bridge
Wotan	Baritone	Air	Valhalla	Rainbow Bridge
Erda	Alto	Wisdom	Middle of Universe	Fremont
Erda	Alto	Fate	Middle of Universe	Fremont
Siegfried	Tenor	Sword	Forest	Hurricane Ridge
Brunnhilde	Soprano	Horse	Rock	Enchanted Fire Ring
Freia	Soprano	Love	Valhalla	Rainbow Bridge
Loge	Tenor	Fire	Valhalla	Rainbow Bridge
Hagen	Bass	Drink	Hut	Seward Park
Getrune	Soprano	Drink	House	Wallingford

(a) This table is not in BCNF. Based only on the data given above, identify the functional dependencies that violate BCNF.

The problematic FDs are:

Character → Voice, Residence, Address
Residence → Address
Address → Residence

There are some others that turned up in some of the answers such as Power, Residence → Voice or Voice, Residence → Character, which were accidents in the data that we missed while proofreading the question. We graded answers that picked up on these accordingly, but the above FDs were sufficient and are the ones that make the most sense in this application (although we didn't expect people to analyze the significance of the data during the test).

Several answers included the notation Residence ↔ Address. We gave credit for this since it was reasonably clear what was meant, but a double-arrow is not standard notation for FDs.

(continued next page)

Question 3 (cont.) (b) Using the functional dependencies you identified in part (a), decompose the table into BCNF. Be sure to indicate which attributes are the key(s) of the various relations, and be sure to indicate any foreign key relationships between the tables.

Be sure to **show the steps** in your decomposition – don't just show the final result. You should also clearly show which functional dependencies you are using at each step in your decomposition, and you should underline the keys in each table so that we can follow your work and evaluate it carefully.

Hint: there may be more than one correct solution to the problem.

The schema of the original table is repeated here to get you started:

RING(Character, Voice, Power, Residence, Address)

There are several possible first steps depending on which FD we pick. We'll use the FD Character → Voice, Residence, Address to get:

T1(Character, Voice, Residence, Address)
T2(Character, Power)

T2 is in BCNF because any two-attribute relation is in BCNF. T1 still has the two bad dependencies Residence → Address and Address → Residence. We can use either one to decompose T1; we'll pick the first one to get

T11(Character, Voice, Residence)
T12(Residence, Address)

where Residence in T11 is a foreign key referring to Residence in T12.

(c) For 1 point of extra credit, and ***only*** if you have time left at the end of the exam, what is the name of Brunnhilde's horse?

Grane

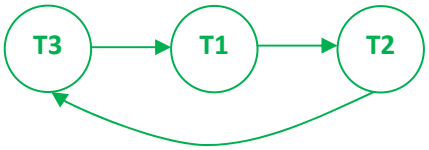
(Most entertaining guess: Hörse)

Question 4. Serialization (18 points). For each of the following schedules,

i. Draw the precedence graph for the schedule.

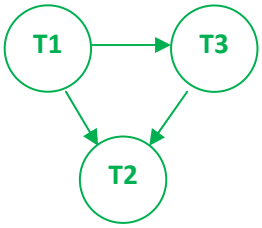
ii. If the schedule is confict-serializable, give one equivalent serial schedule. If the schedule is not conflict-serializable, explain why not.

(a) r1(A) r3(A) w1(A) r2(A) w2(B) w3(B)



This schedule is not conflict serializible because there is a cycle in the precedence graph.

(b) r1(A) w1(A) r3(A) w1(B) r2(A) w3(B) r2(B)



This schedule is conflict-serializable. The only equivalent serial schedule is

r1(A), w1(A), w1(B), r3(A), w3(B), r2(A), r2(B)

Question 5. Two short questions on transactions and logging (18 points)

(a) Two of the choices we have when designing a transaction and buffer manager are whether to use a steal or no-steal policy and whether to use a force or no-force policy. Which combination of these two choices maximizes overall throughput and concurrency in the system? Give a brief explanation for your answer.

Steal, no-force. Steal allows dirty pages to be written to disk even if the corresponding transaction(s) haven't committed, which allows other transactions to reuse buffers. No-force means that dirty pages do not need to be written to disk when a transaction commits. That can allow several transactions to write to the same page without needing separate disk write operations for each one, reducing the time needed for the group of transactions.

(b) In our presentation of the different logging algorithms (undo, redo, and undo-redo), we generally ignored the possibility that a single element (page) in the buffer pool might contain tuples that were being processed concurrently by different transactions. The fact that tuples might share physical pages causes problems for simple undo and redo logging, but not for an undo-redo protocol like ARIES. Briefly, what is (are) the problem(s) that can occur if tuples being processed by different transactions occupy the same page when a simple undo or redo log is used? How does the ARIES protocol solve this (these) problem(s)?

The main problem is that simple undo- and redo-logging have strict requirements about when dirty buffer pages must be written to disk. Undo logging requires that transactions be logged before changes are written, and changes must be written before a transaction can commit. If another transaction has changed a page but not yet logged that change, then writing that page to disk so the first transaction can commit will violate the requirements of undo logging. Redo logging has a similar problem since writes to disk must be delayed until all transactions that have written to a page have committed and have logged both the changes and their commit operations. There is a problem if we try to use a redo log to recover a disk page that contains changes made by both committed and uncommitted transactions. We need to redo the page because of the committed transaction(s), but we can't redo it because of the one(s) that didn't commit.

An undo-redo protocol like ARIES avoids these problems by decoupling the timing of commit operations and buffer writes to disk. As long as all of the log records for an individual transaction are written to stable storage before any corresponding pages are written (WAL – Write Ahead Log), the actual page writes can be done at any time after the log entries, the order and timing of commit operations is independent of other transactions, and the data can be recovered after a crash whether or not the buffers were written to disk.

[Grading note: actual answers were not expected to be this extensive. The main thing we were looking for was an understanding of the issues involved when transactions share buffer pages, and the kinds of conflicts that the simpler logging rules produce.]