Question 1. [12 MARKS]

Consider the following schema for a program that pairs students with mentors who offer career advice.

Relations

- Mentorship(<u>year</u>, sponsor, budget)
  A tuple in this relation represents information about a single year of the mentorship program: the year, the company that sponsored the program, and the budget for the program.
- Mentor(<u>MID</u>, name, email, employer, title, phone)
  A tuple in this relation represents information about a mentor.
- MApplication(<u>MID, year</u>, capacity)
  A tuple in this relation indicates that mentor *MID* applied to be a mentor in a given year and they were willing to mentor up to *capacity* students.
- Student(<u>SID</u>, name, email, phone)
  A tuple in this relation represents information about a student.
- SApplication(<u>SID, year</u>, cgpa)
  A tuple in this relation indicates that a student applied to be mentored in a given year and had the given cgpa at the time.
- Expertise(<u>MID, year, area</u>)
  A tuple represents an area, such as "web development" that a mentor declared expertise in for a particular year.
- Interest(<u>SID, year, area</u>, priority)
  A tuple represents an area that a student has declared an interest in for a particular year. The attribute *priority* indicates how strong the student's interest in that area is, with 1 being the highest priority.
- Match(<u>MID, SID, year</u>)
  A tuple indicates that the mentor *MID* was matched with student *SID* in the given year.

Integrity constraints

The obvious integrity constraints apply:

MApplication[MID] $\subseteq$ Mentor[MID]	MApplication[year] $\subseteq$ Mentorship[year]
SApplication[SID] $\subseteq$ Student[SID]	SApplication[year] $\subseteq$ Mentorship[year]
Expertise[MID] $\subseteq$ Mentor[MID]	Expertise[year] $\subseteq$ Mentorship[year]
Interest[SID] $\subseteq$ Student[SID]	Interest[year] $\subseteq$ Mentorship[year]
Match[MID] $\subseteq$ Mentor[MID]	Match[SID] $\subseteq$ Student[SID]
Match[year] $\subseteq$ Mentorship[year]	

On the next page, write the specified queries in relational algebra, using only the basic operators we used in class and on the assignment: $\Pi, \sigma, \bowtie, \times, \cap, \cup, -, \rho$.:

1. Let's say that a student is hard to match in a given year if they applied that year yet (a) they gave no interests that year or (b) they rated all their interests with the same priorty that year. Write a query in relational algebra to find the SIDs of the students who have applied in at least one year and were hard to match in every year in which they applied.

2. Write a query in relational algebra to find the MID, name, and title of mentors who were matched with two or more students in every year of the mentorship program.

   Note: The schema is repeated on the last page of the exam. You may tear it off for reference.

Question 2. [12 MARKS]

Now write the queries below on the same schema, but in SQL.

1. For each year of the program, report the year, the number of mentor-student matches, the number of mentors who were not matched with a student, and the largest number of mentees that any mentor was matched with. Where a value is zero, for example if there were no unmatched mentors in a year, you may show either zero or `null`. Produce a table with the following form:

```
 year | totalmatches | unmatched | mostmentees
------+--------------+-----------+-------------
```

Solutions:

Not very concise, but works:

```
create view matchcounts as
select m.year, m.mid, count(match.sid) nstudents
from mapplication m left join match using (mid, year)
group by m.year, m.mid;

create view mostmenteesperyear as
select year, max(nstudents) as mostmentees
from mentorship left join matchcounts using (year) group by year;

create view unmatchedcountsperyear as
select year, count(*) as unmatched
from (
   (select mid, year
   from mapplication)
   except
   (select mid,year
   from match)
) as unmatched
group by year;

create view matchesperyear as
select year, count(mid) as totalmatches
from mentorship left join match using (year) group by year;

select *
from matchesperyear left join unmatchedcountsperyear using(year)
join mostmenteesperyear using(year);
```

2. For each year of the program, and for each cgpa range, report the number of matched students. The cgpa ranges are: "hi" for 3.4 and above, "middle" for at least 1.7 but below 3.4, and "lower" for below 1.7. Produce a table with the following form:

```
 year | range | count
------+-------+-------
```

If a year had no matched students in a certain range, or even no matched students at all, you may either show a count of 0, a count of `null` or omit the tuple entirely.

Solutions:

Not very concise, but works:

```
create view high as
select year, cast('high' as text) as range, count(*)
from match join sapplication using(sid, year) where cgpa >= 3.4
group by year;

create view med as
select year, cast('med' as text) as range, count(*)
from match join sapplication using(sid, year) where cgpa >=1.7 and cgpa < 3.4
group by year;

create view low as
select year, cast('low' as text) as range, count(*)
from match join sapplication using(sid, year) where cgpa < 1.7
group by year;

(select * from high) union (select * from med) union (select * from low);
```

Question 3. [7 MARKS]

Consider the following table definitions.

```
CREATE TABLE T (
   M INT PRIMARY KEY,
   N INT,
   O INT UNIQUE
);

CREATE TABLE S (
   P INT PRIMARY KEY,
   Q INT,
   FOREIGN KEY (Q) REFERENCES T(M) ON UPDATE SET NULL ON DELETE CASCADE
);

CREATE TABLE R (
   A INT PRIMARY KEY,
   B INT,
   C INT,
   FOREIGN KEY (C) REFERENCES S(P) ON DELETE SET NULL,
   FOREIGN KEY (A) REFERENCES T(O) ON DELETE CASCADE
);
```

Suppose the tables have been populated as follows:

R:

a	b	c
6	2	22
9	2	1
3	1	2

S:

p	q
7	1
8	4
0	7
22	1
1	7
18	5
2	5

T:

m	n	o
1	2	13
4	5	6
7	8	9
5	5	3

1. Write a query to change table `T` so that every tuple with `m=4` (in this case, there's only one) has that value changed to 9.

   Solution:

   ```
   update T set M = 9 where M = 4;
   ```

2. Modify the data below to show the contents of the three tables after your query is executed.

```
R:
 a | b | c
---+---+----
 6 | 2 | 22
 9 | 2 |  1
 3 | 1 |  2
```

```
S:
 p  | q
----+---
  7 | 1
  8 | 4
  0 | 7
 22 | 1
  1 | 7
 18 | 5
  2 | 5
```

```
T:
 m | n | o
---+---+----
 1 | 2 | 13
 4 | 5 |  6
 7 | 8 |  9
 5 | 5 |  3
```

Solution:

```
csc343h-dianeh=> update T set M = 9 where M = 4;
UPDATE 1
csc343h-dianeh=> select * from R;
 a | b | c
---+---+----
 6 | 2 | 22
 9 | 2 |  1
 3 | 1 |  2
(3 rows)

csc343h-dianeh=> select * from S;
 p  | q
----+---
  7 | 1
  0 | 7
 22 | 1
  1 | 7
 18 | 5
  2 | 5
  8 |
(7 rows)

csc343h-dianeh=> select * from T;
 m | n | o
---+---+----
 1 | 2 | 13
 7 | 8 |  9
 5 | 5 |  3
 9 | 5 |  6
(4 rows)
```

3. Suppose, instead, we were to start with the original tables and delete the tuple from `T` whose value for `m` is 7. Show the contents of the three relations after these changes.

```
R:
 a | b | c
---+---+----
 6 | 2 | 22
 9 | 2 |  1
 3 | 1 |  2
```

```
S:
 p  | q
----+---
  7 | 1
  8 | 4
  0 | 7
 22 | 1
  1 | 7
 18 | 5
  2 | 5
```

```
T:
 m | n | o
---+---+----
 1 | 2 | 13
 4 | 5 |  6
 7 | 8 |  9
 5 | 5 |  3
```

Solution:

```
csc343h-dianeh=> delete from T where M = 7;
DELETE 1
csc343h-dianeh=> select * from R;
 a | b | c
---+---+----
 6 | 2 | 22
 3 | 1 |  2
(2 rows)

csc343h-dianeh=> select * from S;
 p  | q
----+---
  7 | 1
  8 | 4
 22 | 1
 18 | 5
  2 | 5
(5 rows)

csc343h-dianeh=> select * from T;
 m | n | o
---+---+----
 1 | 2 | 13
 4 | 5 |  6
 5 | 5 |  3
(3 rows)
```

4. Suppose, instead, we were to start with the original tables and delete the tuple from `T` whose value for `m` is 5. Show the contents of the three relations after these changes.

```
R:
 a | b | c
---+---+----
 6 | 2 | 22
 9 | 2 |  1
 3 | 1 |  2
```

```
S:
 p  | q
----+---
  7 | 1
  8 | 4
  0 | 7
 22 | 1
  1 | 7
 18 | 5
  2 | 5
```

```
T:
 m | n | o
---+---+----
 1 | 2 | 13
 4 | 5 |  6
 7 | 8 |  9
 5 | 5 |  3
```

Solution:

```
csc343h-dianeh=> delete from T where M = 5;
DELETE 1
csc343h-dianeh=> select * from R;
 a | b | c
---+---+----
 6 | 2 | 22
 9 | 2 |  1
(2 rows)

csc343h-dianeh=> select * from S;
 p  | q
----+---
  7 | 1
  8 | 4
  0 | 7
 22 | 1
  1 | 7
(5 rows)

csc343h-dianeh=> select * from T;
 m | n | o
---+---+----
 1 | 2 | 13
 4 | 5 |  6
 7 | 8 |  9
(3 rows)
```

Question 4. [5 MARKS]

Part (a) [2 MARKS]

Briefly describe two important advantages of using a PreparedStatement rather than a Statement in JDBC.

1.

2.

Part (b) [1 MARK]

Which of the following is true in SQL? Circle one.

1. If a boolean condition mentions an attribute whose value is NULL, it evaluates to UNKNOWN. The value UNKNOWN satisfies a CHECK constraint but not a WHERE clause.
2. If a boolean condition mentions an attribute whose value is NULL, it evaluates to UNKNOWN. The value UNKNOWN satisfies a WHERE clause but not a CHECK constraint.
3. If a boolean condition mentions an attribute whose value is NULL, it evaluates to FALSE. The value FALSE satisfies a CHECK constraint but not a WHERE clause.
4. If a boolean condition mentions an attribute whose value is NULL, it evaluates to FALSE. The value FALSE satisfies a WHERE clause but not a CHECK constraint.
5. None of the above.

Part (c) [2 MARKS]

Consider a relation R on attributes ADEF. Are these two sets of functional dependencies equivalent: $S_1 = \{AF \to DE\}$ and $S_2 = \{A \to DE, F \to DE\}$? Circle one answer.

EQUIVALENT NOT EQUIVALENT

If equivalent, explain. If not, give a counterexample: an instance of R that satisfies one set of FDs but not the other, and explain which set of FDs your instance satisfies.

xxx

Question 5. [6 MARKS]

Part (a) [4 MARKS]

Suppose the following XML document is valid with respect to its DTD:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE DATA SYSTEM "blah.dtd">
<DATA>
   <ANIMALS>
      <COW name = "Snowball" home = "Red Barn Acres"/>
      <HEN name = "Henrietta" birthdate = "120724" home = "Red Barn Acres"/>
   </ANIMALS>
   <FARMS>
      <FARM owner = "Old MacDonald" name = "Red Barn Acres">
         A picturesque hobby farm on 150 acres
      </FARM>
      <FARM owner = "Egg Masters Inc" name = "Farm 23">
         A factory farm on a quarter section of land
      </FARM>
   </FARMS>
</DATA>
```

For each of the following rules, circle Yes or No to indicate whether it could be part of the DTD.

`<!ELEMENT DATA (ANIMALS+, FARMS*)>`	Yes	No
`<!ATTLIST COW birthdate CDATA #IMPLIED>`	Yes	No
`<!ELEMENT HEN EMPTY>`	Yes	No
`<!ELEMENT ANIMALS (HEN | COW)*>`	Yes	No

Part (b) [1 MARK]

Write a DTD definition for element FARMS that accepts the above instance document and enforces this rule: There must be at least two farms in the file. If this is not possible, explain why

Solution:

```
<!ELEMENT FARMS (FARM, FARM+)>
```

Part (c) [1 MARK]

Write a DTD definition for attribute `name` of element `FARM` that accepts the above instance document and enforces this rule: No two farms have the same name. If this is not possible, explain why.

Solution:

It's not possible because blanks are not allowed in an ID attribute.

Question 6. [15 MARKS]

Suppose the following XML file is called `another.xml`.

```
<?xml version="1.0" standalone="yes"?>
<a p="hello">
  <b x="1" y="5"/>
  <c n="100">
     <d real="true">no way</d>
     <d real="false">yes way</d>
     <d real="false">possibly</d>
  </c>
  <b x="3" y="1"/>
  <b x="2" y="6"/>
  <c n="52">
     <d real="false">truly</d>
  </c>
  <c n="50">
     <d real="true">really</d>
     <d real="true">actually</d>
  </c>
</a>
```

Each of the queries in this question runs without errors. Show the output. We will not be grading the whitespace in your answer, so format it as you wish.

1.
```
let $d := fn:doc("another.xml")
return ($d/a/c[2], $d/a[2], $d/a/b[2])
```

   Solution:

```
<c n="52">
     <d real="false">truly</d>
</c>,
<b x="3" y="1"/>
```

2.
```
let $d := fn:doc("another.xml")
return
   if ($d/a/b/@x = $d/a/b/@y) then (count($d//c)) else (<answer>$d/b[@x>3]</answer>)
```
Solution:

```
3
```

3.
```
<whole> {
   let $d := fn:doc("another.xml")
   for $x in $d//d
   where $x/@real
   return
      <one> { $x/parent::c/@n } </one>
} </whole>
```
Solution:

```
<whole>
      <one n="100"/>
      <one n="100"/>
      <one n="100"/>
      <one n="52"/>
      <one n="50"/>
      <one n="50"/>
</whole>
```

4.
```
for $thingee in fn:doc("another.xml")//b[@x<@y]/@y
for $thingama in fn:doc("another.xml")/a/c
where $thingama[count(d) > 1]
return
<item arg1 = '{$thingama/@n}' arg2 = '{$thingee}' />
```
Solution:

```
<item arg1="100" arg2="5"/>,
<item arg1="50" arg2="5"/>,
<item arg1="100" arg2="6"/>,
<item arg1="50" arg2="6"/>
```

5.
```
let $d := fn:doc("another.xml")
let $thing := $d/a/c
for $other in $thing/d[. > "q"]
return <answer>{$other}</answer>
```
Solution:

```
<answer><d real="false">yes way</d></answer>,
<answer><d real="false">truly</d></answer>,
<answer><d real="true">really</d></answer>
```

Question 7. [13 MARKS]

Suppose we have a valid XML file called `recipes.xml` that begins with this DTD:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE RECIPES [
   <!ELEMENT RECIPES (RECIPE)+>
   <!ELEMENT RECIPE (INGREDIENTS, STEPS)>
   <!ATTLIST RECIPE name CDATA #REQUIRED>
   <!ATTLIST RECIPE type CDATA #IMPLIED>
   <!ATTLIST RECIPE keywords CDATA #IMPLIED>
   <!ELEMENT INGREDIENTS (INGREDIENT)+>
   <!ELEMENT INGREDIENT (NAME, QUANTITY)>
   <!ELEMENT NAME (#PCDATA)>
   <!ELEMENT QUANTITY EMPTY>
   <!ATTLIST QUANTITY amount CDATA #REQUIRED>
   <!ATTLIST QUANTITY units CDATA #REQUIRED>
   <!ELEMENT STEPS (STEP)+>
   <!ELEMENT STEP (#PCDATA)>
]>
```

Part (a) [1 MARK]

Complete the file by writing valid XML that has one `RECIPE` with one `INGREDIENT`, and one `STEP`.

Solution:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE RECIPES SYSTEM "recipes.dtd">
<RECIPES>
   <RECIPE name="pumpkin pie" type="dessert">
      <INGREDIENTS>
         <INGREDIENT>
             <NAME>olive oil</NAME>
             <QUANTITY amount="10" units = "ml"/>
         </INGREDIENT>
      </INGREDIENTS>
      <STEPS>
         <STEP>
            Stir together flour and salt.
         </STEP>
      </STEPS>
   </RECIPE>
</RECIPES>
```

Part (b) [5 MARKS]

Write a query in XQuery to return all pairs consisting of the name of a recipe whose type is "main" and the name of a recipe whose type is "dessert", where both contain the keyword "low-fat" and neither one has more than five steps.

The format of your output does not matter. The function `contains(s1, s2)` returns true iff string `s1` contains string `s2`.

Solution:

```
let $d := fn:doc("recipes.xml")
for $recipe1 in $d//RECIPE
for $recipe2 in $d//RECIPE
where
$recipe1/@type = "main" and
$recipe2/@type = "dessert" and
fn:contains($recipe1/@keywords, "low-fat") and
fn:contains($recipe2/@keywords, "low-fat") and
count($recipe1//STEP) < 5 and count($recipe2//STEP) < 5
return ($recipe1/@name, $recipe2/@name)
```

Output on one sample file:

```
greywolf% galax-run part1.xq
attribute name {"tea"}, attribute name {"pumpkin pie"}
```

For reference, here is the DTD again:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE RECIPES [
   <!ELEMENT RECIPES (RECIPE)+>
   <!ELEMENT RECIPE (INGREDIENTS, STEPS)>
   <!ATTLIST RECIPE name CDATA #REQUIRED>
   <!ATTLIST RECIPE type CDATA #IMPLIED>
   <!ATTLIST RECIPE keywords CDATA #IMPLIED>
   <!ELEMENT INGREDIENTS (INGREDIENT)+>
   <!ELEMENT INGREDIENT (NAME, QUANTITY)>
   <!ELEMENT NAME (#PCDATA)>
   <!ELEMENT QUANTITY EMPTY>
   <!ATTLIST QUANTITY amount CDATA #REQUIRED>
   <!ATTLIST QUANTITY units CDATA #REQUIRED>
   <!ELEMENT STEPS (STEP)+>
   <!ELEMENT STEP (#PCDATA)>
]>
```

Suppose we also have this DTD for shopping lists:

```
<!ELEMENT LISTS (SHOPPINGLIST)*>
<!ELEMENT SHOPPINGLIST (ITEM)+>
<!ATTLIST SHOPPINGLIST recipe CDATA #REQUIRED>
<!ELEMENT ITEM EMPTY>
<!ATTLIST ITEM name CDATA #REQUIRED>
```

Part (c) [7 MARKS]

Write a query in XQuery to return XML containing the following information for each recipe in file `recipes.xml` whose type is "dessert": the recipe name and a list of the names of all the ingredients in the recipe. The xml generated must conform to the DTD defined in file `shop.dtd`. Generate only the xml elements, not the "preamble" (the part that declares the xml version etc.).

Solution:

```
<LISTS> {
   let $doc := doc("recipes.xml")
   for $r in $doc//RECIPE[@type = "dessert"]
   let $items :=
      for $it in $r//INGREDIENT/NAME
      return <ITEM name="{data($it)}"/>
   return <SHOPPINGLIST recipe="{$r/@name}">{$items}</SHOPPINGLIST>
} </LISTS>
```

Output on one sample file:

```
greywolf% galax-run part2.xq
<LISTS>
   <SHOPPINGLIST recipe="pumpkin pie">
      <ITEM name="olive oil"/>
      <ITEM name="butter"/>
      <ITEM name="olive oil"/>
      <ITEM name="butter"/>
   </SHOPPINGLIST>
   <SHOPPINGLIST recipe="coconut cream pie">
      <ITEM name="butter"/>
      <ITEM name="bacon"/>
      <ITEM name="oranges"/>
   </SHOPPINGLIST>
</LISTS>
```

Question 8. [8 MARKS]

Consider relation $R(A, B, C, D, E, F)$ with functional dependencies S.

$$S = \{ABCD \to F, \quad ABCE \to D, \quad BC \to EF, \quad C \to B, \quad DEF \to C, \quad DF \to AE\}$$

Part (a) [4 MARKS]

Compute a minimal basis for S.

Part (b) [2 MARKS]

Compute all keys for R.
Suggestion: Work from your minimal basis rather than the original set of functional dependencies.

Solution:

Part (c) [2 MARKS]

Employ the 3NF synthesis algorithm to obtain a lossless and dependency-preserving decomposition of relation R into a collection of relations that are in 3NF.

Solution:

(Abbreviated)

```
Starting FDs:
A B C D -> F
A B C E -> D
B C -> E F
C -> B
D E F -> C
D F -> A E

First attempt to simplify:

    Don't need A B C D -> F

    Removed E from LHS of D E F -> C

    Removed E from LHS of A B C E -> D

    Removed B from LHS of B C -> F

    Removed B from LHS of B C -> E

Second attempt to simplify:

    Don't need D F -> E
```

Question 9. [8 MARKS]

Part (a) [4 MARKS]

Suppose we have the relation R(A, B, C, D, E) with functional dependencies $C \rightarrow AE$, and $D \rightarrow C$. Use the chase test to determine whether the decompositions below are lossless join decompositions.

Would a decomposition into $R1(A, D, E)$ and $R2(B, C, D)$ be lossless? YES NO

Would a decomposition into $R1(A, C, D)$ and $R2(B, D, E)$ be lossless? YES NO

Part (b) [4 MARKS]

Consider the relation R on attributes $ABCDEFG$, with the following functional dependencies:

$$A \rightarrow CF, \quad BCG \rightarrow D, \quad CF \rightarrow AH, \quad D \rightarrow B, \quad H \rightarrow DEG$$

Above, circle each dependency that violates BCNF. Then show below what attributes would go into the two relations that would result from the first step of BCNF decomposition on R. (There may be several correct answers.) Do not show the FDs that apply.

Rough work:

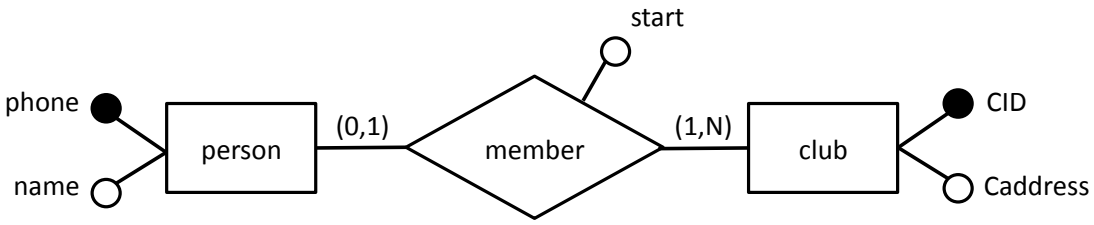
One new relation would have attributes:

The other new relation would have attributes:

Solutions:

Question 10. [8 MARKS]

Consider the following ER diagram:



Below are several possible translations of this ER diagram into a schema. For each, indicate whether it is a good translation: one that represents all the information as described in the ER diagram, does not allow redundancy in the data, and does not invite unnecessary null values.

1. Schema:
   Person(name, <u>phone</u>, CID, start, Caddress)

   Is it a good translation? YES NO

   Explain:

2. Schema:
   Person(name, <u>phone</u>, CID, start)

   Club(<u>CID</u>, Caddress)

   Is it a good translation? YES NO

   Explain:

3. Schema:
   Person(name, <u>phone</u>)

   Club(<u>CID, MID</u>, Caddress, start)

   Is it a good translation? YES NO

   Explain:

4. Schema:
   Person(name, <u>phone</u>)

   Club(<u>CID</u>, Caddress)

   Member(<u>phone, CID</u>, start)

   Is it a good translation? YES NO

   Explain:

Question 11. [10 MARKS]

Design an Entity-Relationship Model (ER Diagram) for the information below about a company that delivers goods via a fleet of trucks. Clearly indicate primary keys by using solid circles. Use a pair of the form (minimum, maximum) to show the cardinalities with which an entity participates in a relationship.

- A truck has a unique license plate number, a type (for instance "cube van" or "18-wheeler", and a capacity, which includes both the volume of goods it can carry, and the maximum weight of goods it can carry.
- A fleet is a group of 1 or more trucks. It has a depot name and a city, and the two together are unique. Each truck belongs to exactly one fleet.
- A driver has a name, date of birth, and unique driver's license number.
- An item for delivery has a unique item ID, and a description. We will also record whether or not the item needs refridgeration.
- A trip involves a driver taking a truck from a source location to a destination, on a given departure date, with a load of items on the truck.
- A load has a unique load ID and a date when it was loaded. There is at least one item in every load. An item may not be part of any load (for instance, if the item has not even been scheduled to be delivered). An item may also be part of several loads (for instance, if it is delivered on several occassions to different destinations).
- We may have a record of any number of trips for a driver, even zero (for instance if he or she is a new employee). Similarly, we may have a record of any number of trips for a truck, even zero (for instance if it is a new truck).
- A load may or may not have gone on a trip, but it cannot go on more than one trip. (If the same items are delivered another time, we consider that a different trip.)

Don't try to express any constraints other than those stated above.

Put your ER diagram on the next page.

ER diagram:

Solution:

Below is a copy of the schema used in questions 1 and 2. Nothing on this page will be graded. You may tear this page off for reference.

Relations

- Mentorship(year, sponsor, budget)
  A tuple in this relation represents information about a single year of the mentorship program: the year, the company that sponsored the program, and the budget for the program.
- Mentor(MID, name, email, employer, title, phone)
  A tuple in this relation represents information about a mentor.
- MApplication(MID, year, capacity)
  A tuple in this relation indicates that mentor *MID* applied to be a mentor in a given year and they were willing to mentor up to *capacity* students.
- Student(SID, name, email, phone)
  A tuple in this relation represents information about a student.
- SApplication(SID, year, cgpa)
  A tuple in this relation indicates that a student applied to be mentored in a given year and had the given cgpa at the time.
- Expertise(MID, year, area)
  A tuple represents an area, such as "web development" that a mentor declared expertise in for a particular year.
- Interest(SID, year, area, priority)
  A tuple represents an area that a student has declared an interest in for a particular year. The attribute *priority* indicates how strong the student's interest in that area is, with 1 being the highest priority.
- Match(MID, SID, year)
  A tuple indicates that the mentor *MID* was matched with student *SID* in the given year.

Integrity constraints

The obvious integrity constraints apply:

MApplication[MID] $\subseteq$ Mentor[MID]
MApplication[year] $\subseteq$ Mentorship[year]
SApplication[SID] $\subseteq$ Student[SID]
SApplication[year] $\subseteq$ Mentorship[year]
Expertise[MID] $\subseteq$ Mentor[MID]
Expertise[year] $\subseteq$ Mentorship[year]
Interest[SID] $\subseteq$ Student[SID]
Interest[year] $\subseteq$ Mentorship[year]
Match[MID] $\subseteq$ Mentor[MID]
Match[SID] $\subseteq$ Student[SID]
Match[year] $\subseteq$ Mentorship[year]