University of Waterloo: Midterm Examination **Winter Term 2017**

Student Name: _____

Student Signature: _____

Student Identification Number: _____

E&CE 356	Database Systems
Section: 001	Instructor: Paul Ward
Monday, February 13th 2017	7:00 PM to 8:30 PM
Duration of Exam:	1.5 hours
Number of Exam Pages (including cover sheet):	10 pages / 4 questions
Exam Type:	Closed Book

General Notes

1. No electronic devices including no calculators.
2. Read the entire questions *carefully* before answering.
3. State any assumptions clearly and whenever needed.
4. No questions permitted. When in doubt, make an assumption, write it down.
5. Some SQL reminders are listed on Page 10.
6. *aquila non captat muscas*

Question	**Maximum**	**Score**						**Total**
1	30	(a)	(b)	(c)	(d)	(e)	(f)	
2	30	(a)	(b)	(c)	(d)	(e)		
3	30	(a)	(b)	(c)		(d)		
3	10	(a)			(b)			
Total	100							

Sample solutions in red; different solutions may also work, so do not assume because your solution differs it is necessarily wrong.

Database for the midterm

The following schema and data will be used for the questions on this midterm. It is a small database for a vet, providing details about pet owners, which pets they own, and some details about the pets. The table name (relation) is in **bold** and the primary key is underlined.

Owner		
OwnerID int	Name varchar(10)	Phone char(8)
123	Bart	123-4567
456	Lisa	789-1011
789	Maggie	722-2222
101	Homer	123-4567
112	Maggie	123-4567

Owns		
OwnerID int	PetID int	Registered date
123	1	11/11/2011
123	4	14/12/2013
456	2	21/03/2009
789	3	02/07/2015
123	5	NULL

Pet					
PetID int	Name varchar(10)	Type varchar(10)	SubType varchar(10)	BirthDate date	Weight int
1	Rainbow	Dog	Beagle	2011-10-10	35
2	Loki	Cat	Siamese	2013-08-17	12
3	Smeagol	Dog	Labrador	2014-03-01	53
4	Ploppy	Dog	Poodle	2012-11-24	13
5	Reggie	Bird	Parrot	2008-05-11	1

PetTypes
Type varchar(10)
Dog
Cat
Bird

The relation "**Owner**" has OwnerID as its primary key and no foreign keys.

The relation "**PetType**" has a single column, Type, which is its primary key, and identifies all possible pet Types.

The relation "**Pet**" has PetID has its primary key. The attribute "Type" is a foreign key that references relation "**PetType**", attribute "Type".

The relation "**Owns**" has PetID as its primary key. The "PetID" is a foreign key, referencing relation **Pet**, attribute "PetID". The relation also records when a pet was first registered with the vet.

If it helps, the specific table creation commands for these relations as SQL tables are:

```
create table PetType(Type char(6), primary key(Type));

create table Owner(OwnerID int primary key, Name varchar(10), Phone varchar(10));

create table Pet(PetID int primary key, Name varchar(10), Type varchar(10),
                 SubType varchar(10), BirthDate Date, Weight int,
                foreign key(Type) references PetType(Type));

create table Owns(OwnerID int, PetID int primary key, Registered Date,
                  foreign key(OwnerID) references Owner(OwnerID),
                  foreign key(PetID) references Pet(PetID));
```

1. [30 marks] SQL Comprehension: Considering the database schema and the particular data on page 2, for each of the following queries identify
(i) in a single sentence, what is the query computing?
(ii) what is the output for this particular dataset?

(a) `select Name,Weight from Pet where Type='Dog' order by name;`

What are the names and weights of all dogs, sorted by name?

Name	Weight
Ploppy	13
Rainbow	35
Smeagol	53

(b) `select Type,count(distinct PetID) as Num from Pet group by Type;`

How many pets are there of each type?

Type	Num
Bird	1
Cat	1
Dog	3

(c) `select count(PetID)/count(distinct OwnerID) as AO from Owns;`

How many pets does each owner own on average?

AO
1.6667

(d)
```
select O.Name,sum(Weight) as TPW from Owner as O natural join Owns
    inner join Pet as P using (PetID) group by O.Name order by TPW desc;
```

What is the total weight of all pets owned by each owner, ordered in decreasing order of total weight

Name	TPW
Maggie	53
Bart	49
Lisa	12

(e)
```
select O.Name,P.Name,P.BirthDate from Owner as O inner join Owns using (OwnerID)
    inner join Pet as P using (PetID) where Type='Dog' and not exists
     (select * from Pet as Q where P.BirthDate > Q.BirthDate and Type = 'Dog');
```

What is the name of the owner, the name of the dog, and the birthdate of the oldest dog? (There will be multiple rows if there is more than one oldest dog.)

Name	Name	BirthDate
Bart	Rainbow	2011-10-10

(f)
```
select Type, AvgWt from (select Type,A.AW,avg(weight) as AvgWt from Pet,
    (select avg(weight) as AW from Pet) as A group by Type,A.AW having AvgWt < A.AW) as B;
```

Which pet types have average weight below the average weight of all pets, and what is their average weight?

Type	AvgWt
Bird	1.0000
Cat	12.0000

2. [30 marks] Query Creation: Considering the database schema and the particular data on page 2, solve the following queries with *either* Relational Algebra *or* SQL.

(a) What is the average weight of dogs?

```
select avg(weight) from Pet where Type='Dog';
```

(b) What is the average weight of each Type of Pet, ordered by increasing weight?

```
select Type,avg(weight) from Pet group by Type order by avg(weight);
```

(c) Who are the owners of the heaviest pets, by Type? You should list the owner's Name and the pet's Type.

```
select O.Name, P.Type from Owner as O inner join Owns using(OwnerID)
                                      inner join Pet as P using(PetID)
      where not exists
        (select * from Pet as B where B.Type=P.Type and B.Weight > P.Weight);
```

(d) What is the average number of people per pet in each household? A household is determined by the phone number: two people (Owners) with the same phone number are deemed to be in the same household; those with different phone numbers are deemed to be in different households.

The straightforward solution is to compute the PeoplePerHouse, the PetsPerHouse, and then join them on the phone number:

```
select avg(PersonCount/PetCount) from
   (select Phone,count(distinct OwnerID) as PersonCount from Owner group by Phone) as PPH
inner join
   (select Phone,count(P.PetID) as PetCount from Owner as O inner join Owns as P
            using(OwnerID) group by Phone) as PetsPerHouse
       using (Phone);
```

Note: The first subselect counts People Per House (PPH) and will be completely correct even if a house has no pets. The second subselect counts Pets Per House but will omit any house without pets (*i.e.*, there is an owner in the Owner table, but no associated pet owned). If we want to correct for this, that needs to be changed to:

```
select Phone,count(P.PetID) as PetCount from Owner as O left outer join Owns as P
       using(OwnerID) group by Phone
```

However, this is not sufficient, since it results in houses with a PetCount of 0 and PersonCount/PetCount is then NULL, and therefore gets ignored in the average calculation. To correct for this, you would need the main select to be written as:

```
select 1/avg(PetCount/PersonCount) from ...
```

An alternate solution is:

```
select avg(PPP) from
   (select count(distinct Phone,OwnerID)/count(distinct PetID) as PPP
   from Owner left outer join Owns using (OwnerID) group by Phone) as B;
```

Note 1: an inner join is insufficient, since it ignores the people in the Owner table in the house who are not the identified pet Owner (*e.g.*, Homer and Maggie(112) are in the same home as Bart, but not identified as the pet owner.

Note 2: As with the first solution, this ignores the issue of houses where there are no pets; to address that a full outer join is needed.

(e) What are the names and phone numbers of pet owners who own every type of pet?

$(\Pi_{O.Name,Phone,Type}(\rho_O(Owner) \bowtie_{OwnerID} Owns \bowtie_{PetID} Pet)) \div PetType$

or in SQL, create a view for "r" and do the division: $r \div s = \Pi_{R-S}(r) - (\Pi_{R-S}((\Pi_{R-S}(r) \times s) - r))$ ("s" is PetType):

```
create view R as select Name,Phone,Type from
       (select O.Name,Phone,P.Type from Owner as O inner join Owns using (OwnerID)
                                inner join Pet as P using (PetID)) as TMP;


select distinct Name,Phone from R where (Name,Phone) not in
       (select Name,Phone from (select Name,Phone from R) as TMP,PetType
                    where (Name,Phone,Type) not in (select * from R));
```

An alternate solution based on the number of pet types is:

```
select Owner.Name,Phone from Pet natural join Owns inner join Owner using (OwnerID)
   group by OwnerID having (count(distinct Type)) = (select count(Type) from PetType);
```

3. [30 marks] Normalization: Considering the database schema on page 2, in addition to the functional dependencies implied by the primary keys, the following functional dependency exists: SubType → Type

(a) What relations, if any, need to be changed to make the schema Third-Normal Form? If changes are required, identify them, including any new relations, and any changes to primary and foreign keys.

The **Pet** relation is not in 3NF because SubType is not a superkey and Type is not in any candidate key (the only candidate key is the PetID).

The **Pet** relation needs to be broken into two relations: **R1**: (SubType,Type) and **R2**: (PetID, Name,SubType,BirthDate,Weight).

SubType must be primary key for **R1**. Type must be be foreign key to **PetTypes**. While it looks like there is little advantage in the separate **PetTypes** table, this would leave a single table (SubType, Type) without a foreign key, and thus there would no longer be a check on whether or not the Type within this table was from a valid set of Pet Types.

PetID must be primary key for **R2**. SubType must be foreign key to **R1**:SubType.

Note that although **Pet** and **Owns** have PetID as their Primary Key, this does not mean they have to be merged into a single relation; it is not required for 3NF; that said, the only thing that being in separate relations means is that it is posible for their to be pets in the **Pet** relation that are not in the **Owns** relation, which in general is not desirable. (The converse is not the case because of the Foreign Key from **Owns** to **Pet**)

(b) After your adjustments from part(a), if any, is the schema now also in BCNF? If any further changes are required, identify them, including any new or changed relations, and any changes to primary and foreign keys.

The adjusted schema is already in BCNF.

(All non-trivial FDs $\alpha \rightarrow \beta$ have α as the primary key in their respective relations.)

(c) We would like the schema to (i) allow owners to have more than one phone number (ii) allow pets to be owned by more than one owner. If the current schema does not support this, identify why not.

The current schema does not support either of these. Item (i) is precluded because the phone number is contained within the **Owner** relation, which has OwnerID as primary key. As such, no single OwnerID can have two (or more) phone numbers.

Item (ii) is precluded because the **Owns** relation has PetID has primary key, and therefore cannot have two (or more) OwnerIDs associated with any single PetID.

(d) If needed, adjust the schema to support the requirements from part(a). If you need to make adjustments, identify any new or changed relations, and any changes to primary and foreign keys.

Note 1: Per the note during the exam, this should state: "adjust the schema to support the requirements from part (c)" (Not part (a)).

Note 2: the changes required for this are completely orthogonal to those required to deal with the (a)/(b) portion of this question, and therefore it does not matter whether the changes are relative to the original schema or to the schema that results from part (b).

Changes for item(i): Since this change requires that an owner may have more than one phone number, OwnerID is no longer sufficient as a candidate key. Likewise, given that phone numbers can be shared, Phone cannot be a candidate key. However, Name is not shared. Therefore we still have the functional dependency: OwnerID $\rightarrow$ Name. Since this is non-trivial and OwnerID is not a superkey, this violates BCNF. It will not violate 3NF if Name is part of a candidate key. However, (OwnerID,Phone) is the only candidate key in this relation, since any possible pet owner could share a name with any other pet owner. As such, it also violates 3NF. Therefore the **Owner** relation must be broken down into **R3**: (OwnerID,Name) and **R4**:(OwnerID,Phone). The primary key of **R3** is OwnerID and there are no foreign keys. The primary key of **R4** is (OwnerID,Phone) and OwnerID is foreign key to **R3**:OwnerID.

Changes for item(ii): PetID alone can no longer be a candidate key for **Owns** because there can be multiple OwnerIDs for any given PetID. The combination (PetID,OwnerID) can act as a candidate key, since it will be unique and will uniquely determine the relation. However, just because a pet is owned by more than one owner does not mean that it will have multiple registration dates. Therefore, we still have the functional dependency: PetID $\rightarrow$ Registered. Since this is non-trivial and PetID is not a superkey, this violates BCNF. It will not violate 3NF if Registered is part of a candidate key. It is not part of (PetID,OwnerID) so the question is whether or not there is some other candidate key that Registered is part of. There are, however, no additional candidate keys, and so it is not 3NF either. The **Owns** relation therefore must be divided into **R5**:(OwnerID,PetID) and **R6**:(PetID, Registered). **R5** will have (OwnerID,PetID) as primary key, and the attributes will individually remain as foreign key to their respective tables as before. **R6** will have PetID as primary key, and PetID must also be foreign key to **Pet**:PetID.

4. [10 marks] Debugging: Considering the database schema on page 2, some queries have been created that are not correct.

(a) We would like the names of all dog owners and so we execute the query:

```
select name from Owner natural join Owns natural join Pet where Type='Dog';
```

What is wrong with this query?

Natural join uses attribute (column) names to determine what is being used for the join. The relation **Owner** has attribute Name for the name of the owner; however, the relation **Pet** uses the attribute Name for the name of the pet. As such, this query will return the name of all dog owners that have the same name as their dog.

A correct query for this is: `select Owner.Name from Owner natural join Owns inner join Pet using (PetID) where Type='Dog';` The Name needs to be fully qualified in the select statement because it is the owner name that is desired, not the pet name. It is not ambiguous in the natural-join case because the natural join has a single Name attribute within its output.

(b) Sometimes pets die and are removed from the Pet relation. This may be the last pet that an Owner had with us. We would like to identify any owners who have no Pets and so we use the query:

```
select name, Phone from Owner where name not in
        (select name from Owner inner join Owns using (OwnerID));
```

What is wrong with this query?

This is a subtle problem, though if you look at the sample data on Page 2 there is an instance of the problem. Specifically, in that dataset this query misses OwnerID 112, Maggie because OwnerID 789 is also called Maggie and OwnerID 789 has a pet in the **Owns** table, where OwnerID 112 has no pet.

The problem is that the query is assuming that names are unique, when they are not; OwnerID is the unique attribute and, while we want the name and phone number, not the OwnerID (presumably to call the individual), we must use the OwnerID to determine if the individual has a pet or not.

A correct query for this is: `select Name, Phone from Owner where OwnerID not in (select OwnerID from Owner inner join Owns using (OwnerID));`

Some SQL Reminders

DDL:

```
create table
drop table [if exists <table>]
alter table <table> add <attribute> <domain>
alter table <table> drop <attribute>
create view <view> as <query expression>

Domain Types: char(n), varchar(n), int, smallint, numeric(p,d)
              real, double precision, float(n), date, time

Integrity     not null, default <V>, primary key (<a1, ..., an>),
Constraints:  unique(<a1, ..., an>),
              foreign key (<am, ..., an>) references (<am', ..., an'>)

Referential   on update <...>, on delete <...>
Actions:      cascade, set null, set default, no action

Check Constraints: check (<predicate>)
```

DML:

```
insert into <table> values (...)
update <table> set <attribute = ...> where <predicate>
delete from <table> where <predicate>
select <a1,...,an> from <t1,...,tn> where <predicate>

Options on selection attributes: distinct, as

Options on selection tables: natural join, inner join, outer join
                             <join type> on <predicate> using <attributes>

Set Operations: union, interset, except (use 'all' to retain duplicates)

Aggregate Functions: avg, min, max, sum, count

Grouping: group by, having, order by

Options on predicate: =, !=, <, >, <=, >=, in, not in, exists, not exists,
                      is null, is not null, between, like <string pattern>
```