University of Waterloo
Midterm Examination
Winter Term 2016

Student Name: _____

Student Signature: _____

Student Identification Number: _____

Course Section/Instructor (Please Circle): 001/Ward

Course Number:	E&CE 356
Course Title:	Database Systems
Sections:	001
Instructors:	Paul Ward
Date of Exam:	Thursday, February 25th 2016
Time Period:	5:35 AM to 6:50 AM
Duration of Exam:	75 minutes
Number of Exam Pages (including cover sheet):	8 pages / 3 questions
Exam Type:	Closed Book
Additional Materials Allowed:	None

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 8.
6. *qui audet adipiscitur*

Question	**Maximum**	**Score**					**Total**
1	25	(a)	(b)	(c)	(d)	(e)	
2	25	(a)	(b)	(c)	(d)	(e)	
3	25						
Total	75						

1. [25 marks] SQL Comprehension: Consider the following relations:

Person				
ID int	FirstName char(20)	LastName char(20)	Sex char(1)	BirthDate date
123	Fred	Jones	M	1/10/1974
456	Mary	Jane	F	23/4/1977
789	Jimbo	Jones	M	16/3/1994
101	Alex	Jane	F	13/9/1996
112	Josh	Jones	M	3/2/2015

Parent	
ParentID int	ChildID int
123	789
456	101
123	101
101	112

(a) What is the following query computing (in one sentence) and what is its output for this dataset:

```
select count(distinct ChildID) from Parent;
```

[2 marks] It is counting the number of children (as defined by the Parent table)

[2 marks]

count(distinct ChildID)
3

(b) What is the following query computing (in one sentence), what is its output for this dataset, and what happens if "distinct" is omitted:

```
select count(distinct ParentID) as Total from Parent inner join Person on ID=ParentID
                                    where Sex='F';
```

[2 marks] It is counting the number of mothers.

[2 marks]

Total
2

[3 marks] If "distinct" is omitted there is no difference on this particular dataset; however, in the general case, it would change from counting mothers to the number of times females have had a child (note that this is not the same as the number of children, since it would not count children identified by their father but not by their mother)

(c) What is the following query computing (in one sentence), and what is its output for this dataset:

```
select count(distinct P1.ParentID) from Parent as P1 inner join Parent as P2
                                    where P1.ChildID=P2.ParentID;
```

[2 marks] It is counting the number of grandparents (as defined by the Parent table)

[2 marks]

count(distinct P1.ParentID)
2

(d) What is the following query computing (in a sentence) and what is its output for this dataset:

```
select P.FirstName, P.LastName, P3.FirstName from Person as P inner join Parent as P1
                                                    on P.ID=P1.ParentID
                                                inner join Parent as P2 on P1.ChildID=P2.ParentID
                                                inner join Person as P3 on P2.ChildID=P3.ID
                                                where P.LastName = P3.LastName;
```

[2 marks] It is identifying the names of grandparents with the same last name as their grandchildren (as defined by the Parent table)

[3 marks]

FirstName	LastName	FirstName
Fred	Jones	Josh

(e) What are plausible primary and foreign keys for these two relations? Justify.

[1 mark] For the Person relation, the primary key is ID and there are no foreign keys.

[2 marks] For the Parent relation, the primary key is the combination of ParentID and ChildID; if only ParentID is used, then a person can have only one child, though a child could have multiple parents; conversely, if only the ChildID is used, then each child could have only one parent.

[2 marks] For the Parent relation, both ParentID and ChildID should be foregin keys, referencing the ID of the Person relation; any parent or child must be a person.

2. [25 marks] Query Creation:

Consider the following relational schema used to describe a car-rental database (primary keys are underlined):
Customer (c_num, c_name, city)
Car (licence, make, model, year)
Pickup (r_num, c_num, licence, fee, date, time, city)
Dropoff (r_num, date, time, city)
Explanation:

- Customer defines a unique customer number, his/her name, and location
- Car defines a unique car, together with information about the vehicle
- Pickup defines any rental contract in which the vehicle has been picked up, when and where, and for how much
- Dropoff contains the corresponding information for the return of a vehicle
- The attributes c_num and licence in Pickup are foreign keys that reference the primary keys of Customer and Car, respectively. Likewise, r_num in Dropoff is a foreign key that references the primary key in Pickup.

(a) Using SQL or relational algebra, find the names of customers who have rented a car at least five times in the last year.

[5 marks]
SELECT c_name FROM Customer INNER JOIN Pickup using(c_num) WHERE date > date('2015-02-25') GROUP BY Customer.c_num HAVING COUNT(*) >= 5;

If you prefer, join Customer with a query that does the necessary selection on Pickup, where the join with Customer is solely necessary to get the customer name:

SELECT c_name FROM Customer NATURAL JOIN (SELECT c_num,count(r_num) AS Times FROM Pickup WHERE date > date('2015-02-25') GROUP BY c_num) as P WHERE P.Times >= 5;

You can make the subquery into a view if you like:

CREATE VIEW RentCount as SELECT c_num,count(r_num) AS Times FROM Pickup WHERE date > date('2015-02-25') GROUP BY c_num;
SELECT c_name FROM Customer NATURAL JOIN RentCount WHERE Times >= 5;

(b) Using SQL or relational algebra, identify how many cars are currently out on a rental from Waterloo (*i.e.*, any car that has been picked up in Waterloo but has yet to be dropped off).

[5 marks]
SELECT count(*) FROM Pickup as P where city='Waterloo' and NOT EXISTS (SELECT * FROM Dropoff as D WHERE P.r_num = D.r_num);

If you prefer:

SELECT count(*) FROM Pickup LEFT JOIN Dropoff USING (r_num) WHERE Dropoff.date is NULL and Pickup.city='Waterloo';

(c) Using SQL or relational algebra, identify how many cars are dropped off in the same city in which they are picked up and how many are dropped off in a different city.

[4 marks] SELECT count(*) FROM Pickup INNER JOIN Dropoff using(r_num, city);

[1 mark] SELECT count(*) FROM Pickup INNER JOIN Dropoff using(r_num) where Pickup.city ! = Dropoff.city;

(d) Using SQL or relational algebra, identify the models of cars, by make, that generate the highest revenue (*i.e.* the sum of their fees is the highest).

[5 marks] This is most easily done by creating a Revenue view, thus:
CREATE VIEW Revenue as SELECT make,model,sum(fee) as rev FROM Car NATURAL JOIN Pickup GROUP BY make,model;
and then selecting the model with the maximum revenue within that:
SELECT make,model FROM Revenue WHERE (make,rev) in (SELECT make,max(rev) FROM Revenue group by make);

Note that if you don't create the view, then you need to have the corresponding SELECT statement for that view written twice within the query.

(e) Using SQL or relational algebra, identify the most popular car make and model.

[5 marks] This is similar to the above, though by rental, not by revenue; also, we are looking for the most popular make as well as the most popular model; these can be different. To get the most popular make:

CREATE VIEW PopularMake as SELECT make,count(r_num) as pop FROM Car NATURAL JOIN Pickup GROUP BY make;
and then:
SELECT make FROM PopularMake WHERE (pop) in (SELECT max(pop) FROM PopularMake);

The most popular model is a variant on this (and this is sufficient for full marks on this question):

CREATE VIEW PopularModel as SELECT make,model,count(r_num) as pop FROM Car NATURAL JOIN Pickup GROUP BY make,model;
and then:
select make,model from PopularModel where (pop) in (select max(pop) from PopularModel);

Question: Do we need to have the make as well as the model in the PopularModel table?

3. [25 marks] Functional Decomposition and Normalization: Consider a database that contains information about people living in Canada. A person has a social insurance number (SIN), first name (FirstName), last name (LastName), zero or more middle names (MiddleName1, MiddleName2, ...), and a home address comprising a street number (StreetNumber), a street name (StreetName), a city (City), a province (Province), a postal code (PostCode), and (optionally) an apartment number (AptNumber). In unnormalized form, the Person relation is as follows:

- Person(SIN, FirstName, MiddleName1, MiddleName2, ..., LastName, AptNumber, StreetNumber, StreetName, City, Province, PostCode)

The functional dependencies are as follows:

- SIN $\rightarrow$ FirstName, MiddleName(s), LastName
- SIN $\rightarrow$ AptNumber, StreetNumber, StreetName, City, Province, PostCode
- StreetNumber, StreetName, City, Province $\rightarrow$ PostCode
- PostCode $\rightarrow$ City, Province

Given these functional dependencies, define a BCNF set of relations to capture the "Person" relation. Justify in precise, formal terms (*i.e.*, based on specific functional dependencies), why you have the decomposition that you have. Identify any functional dependencies that will not be enforced by your BCNF relations.

The BCNF relations are as follows:

Person(SIN, FirstName, LastName, StreetNumber, StreetName, PostCode)
MiddleNames(SIN, MiddleName, MiddleNameNumber)
ApartmentNumber(SIN, AptNumber)
PostCode(PostCode,City,Province)

Middle names need a separate relation as there can be zero of them; the additional attribute MiddleNameNumber is needed to identify which middle name is being recorded.

Apartment numbers need a separate relation for the same reason that middle names need a separate relation.

PostCode needs a separate relation because there is a functional dependency "PostCode $\rightarrow$ City, Province" which is not trivial and PostCode is not a superkey of the Person relation.

The functional dependency "StreetNumber, StreetName, City, Province $\rightarrow$ PostCode" is not maintained by this normalization because the LHS is split over two relations.

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>
```