CMPS 182, Final Exam, Spring 2017, Shel Finkelstein

Student Name: Yza Velasco

Student ID: 1674461

UCSC Email: ycvelasc

Final Points

Part	Max Points	Points
I	44	
II	30	
III	32	
Total	106	

The first Section (Part I) of the Spring 2017 CMPS 182 Final is multiple choice and is double-sided. Answer all multiple choice questions on your Scantron sheet. You do not have to hand in the first Section of the Exam, but you must hand in the Scantron sheet, with your name, email and student id on that Scantron sheet. Please be sure to use a #2 pencil to mark your choices on this Section of the Final.

This separate second Section (Parts II and III) of the Final is not multiple choice and is single-sided, so that you have extra space to write your answers. If you use that extra space, please be sure to write the number of the problem that you're solving next to your answer. Please write your name, email and student id on this second Section of the Exam, which you must hand in. You may use any writing implement on this Section of the Exam.

At the end of the Final, please be sure to hand in **both** your Scantron sheet for the first Section of the Exam and also **this second Section of the Exam**, and show your UCSC id when you hand them in.

Part II: (30 points, 6 points each)

Question 23: Assume we have the following tables:

- Persons table, with Primary Key PersonID, which is INTEGER.
- Houses table, with Primary Key HouseID, which also is INTEGER.

These tables have other columns that aren't important for this question.

Here's the Create Statement for another table, the Tenants Table, which gives information about persons who live in houses:

```
CREATE TABLE Tenants
 ( ThePerson INT,
   TheHouse INT,
   Rent DECIMAL(6,2),
   LeaseExpiration DATE,
   PRIMARY KEY(ThePerson, TheHouse)
 );
```

4 digits to the left of point
2 digits after it

Rewrite this CREATE statement (don't do an ALTER) so that ThePerson and TheHouse are Foreign Keys that correspond to the Primary keys of Persons and Houses, respectively. Policies for Referential Integrity should be:

- A tuple in Persons can't be deleted if there are any Tenants tuples referring to that person.
- Deleting a tuple from Houses deletes all Tenants tuples that refer to that house.

Answer 23:

```
CREATE TABLE Tenants (
 ThePerson INT REFERENCES Persons(PersonID),
 TheHouse INT REFERENCES Houses(HouseID) ON DELETE CASCADE,
 Rent DECIMAL(6,2),
 LeaseExpiration DATE,
 PRIMARY KEY (ThePerson, TheHouse)
);
```

Question 24a): In the OLAP Lecture, we discussed Fact tables and Dimension tables. When the 3 tables mentioned in the previous question (Persons, Houses and Tenants) are used together for OLAP, indicate whether each would be a Fact table, a Dimension table or both.

Answer 24a):

FACT - business facts, foreign keys referring to primary keys in dimension tables

DIMENSION - descriptive attributes

Dimension	Persons
Dimension	Houses
FACT	Tenants

Question 24b): Explain clearly (using an example, as well as words explaining the example) what Roll-Up means in OLAP.

Answer 24b):

Roll-up refers to aggregation of a Fact table over one or more of its dimensions.

In question 23, we can do a roll-up over tenants to find the number of persons living in each house, or the average rent of the persons living in each house

Question 25: Assume that you have relations:
Employees(name, age, salary, dept),
Departments(dept, manager)

25a): Write a SQL UPDATE statement that add 1000 to the salary of every employee who is in a department whose manager is 'Washington'. Your answer should just be a single UPDATE statement.

Answer 25a):

```
UPDATE Employees e SET e.salary = e.salary + 1000
WHERE EXISTS
    (SELECT * FROM Departments d
     WHERE e.dept = d.dept
      AND d.manager = 'Washington');
```

25b): Write a SQL DELETE statement that deletes all employees whose age is more than 65.

Answer 25b):

```
DELETE FROM Employees
    WHERE age > 65;
```

Question 26: For each pair of queries, write YES if they are equivalent (always have the same answer) and NO if they are not equivalent (sometimes have different answers). We're using the tables from Question 25 where keys are underlined:

Employees(name, age, salary, dept),
Departments(dept, manager)

26a):

```
SELECT DISTINCT name
FROM Employees;
```

```
SELECT name
FROM Employees;
```

Answer 26a): yes

26b):

```
SELECT d.manager
FROM Departments d
WHERE EXISTS (
    SELECT *
    FROM Employees e
    WHERE d.dept = e.dept
  );
```

```
SELECT d.manager
FROM Department d, Employees e
WHERE d.dept = e.dept;
```

manager appears for every employee in her dept.

Answer 26b): No

26c):

```
SELECT DISTINCT age
FROM Employees
WHERE dept = 'Sales';

UNION ALL

SELECT DISTINCT age
FROM Employees
WHERE dept = 'Marketing';
```

```
SELECT DISTINCT age
FROM Employees
WHERE dept = 'Sales'
      OR dept = 'Marketing';
```

can have duplicates

Answer 26c): NO

Question 27: Suppose that you have a relation:

BookInfo(Author, Citizenship, Title, PageCount)

with the following functional dependencies:

Author → Citizenship

Author, Title → PageCount

27a) Is {Author, Title} a key for BookInfo? Justify your answer fully and clearly.

Answer 27a):

Yes. Using Attribute Closure, we see that {Author, Title}$^+$ is {Author, Title, Citizenship, PageCount}, and that's all the attributes of BookInfo, so {Author, Title} is a superkey for BookInfo.

Is it a key? Yes

{Author}$^+$ is {Author, Citizenship}, and {Title}$^+$ is just {Title}, so no subset of {Author, Title} is a superkey.

27b) Is BookInfo in BCNF? Justify your answer fully and clearly.

Answer 27b):

The functional dependency Author → Citizenship is not trivial, and from 27a, {Author is not a superkey}

No

27c) Is BookInfo in 3NF? Justify your answer fully and clearly.

Answer 27

The functional dependency Author → Citizenship is not trivial, and from 27a, {Author is not a superkey}

Citizenship is not part of any key for BookInfo.

Part III: (32 points, 8 points each)

The familiar relations Drivers, Vehicles and Tickets are shown below, with Primary Keys underlined.

Drivers(LicenseID, Name, Address, BirthDate, EyeColor, HairColor, LicenseExpireDate, LicenseClass)

Vehicles(VIN, Model, Year, VehicleColor, OwnerLicenseID, RegExpireDate, InsuranceCo, InUse)

Tickets(TicketID, VIN, LicenseID, TicketDate, Address, Infraction, Fee, Paid)

Assume that no attributes can be NULL, and that there are no UNIQUE constraints.

The Paid attribute in Tickets is BOOLEAN, just as it was in your Labs.

Write SQL queries for each of the following questions. If you want to create and then use views to answer these questions, that's okay, but views are not required unless a question asks you for them.

Question 28: Find the BirthDate values (with no duplicates) for drivers whose name starts with 'Fra'. In your result, BirthDate values should appear in reverse order, so that later dates come before earlier dates.

Answer 28:

```
SELECT DISTINCT d.birthDate
FROM Drivers d
WHERE d.Name LIKE 'FRA%'
ORDER BY d.birthDate DESC;
```

Question 29: Find the LicenseID, name and address for each driver whose address (in Drivers) isn't the address on any ticket in Tickets.

Answer 29:

```
SELECT d.LicenseID, d.Name, d.address
FROM Drivers d
WHERE d.address NOT IN
    (SELECT t.address FROM Tickets t);
```

Question 30: VehicleColor is 'RED' for red vehicles. For each insurance company, output the number of red vehicles that are insured by that insurance company. Your result should include the insurance company (call that attribute Company) and the number of red vehicles insured by that company (call that attribute RedVehicles). But only include an insurance company in your result if the number of red vehicles insured by that company is greater or equal to 49.

Answer 30:

```
SELECT InsuranceCo AS Company, COUNT(*) AS RedVehicles
FROM Vehicles
WHERE VehicleColor = 'RED'
GROUP BY InsuranceCo
HAVING COUNT(*) >= 49;
```

Question 31: This question has two parts; be sure to answer both.

31a) Create a SQL view called UnpaidTicketFees. For each VIN that appears on at least one unpaid ticket, the UnpaidTicketFees view should give the VIN and the total Fee for the unpaid tickets that have that VIN. In the result of your view, the second attribute should be called UnpaidTotal.

31b) Write a SQL query that outputs the VIN and OwnerLicenseID for vehicles whose UnpaidTotal in the UnpaidTicketFees view is the biggest. Note that there can be more than one vehicle that has the biggest UnpaidTotal.

Answers 31a) and 31b):

a)
```
CREATE VIEW UnpaidTicketFees AS
SELECT t.VIN, SUM(t.Fee) AS UnpaidTotal
FROM Tickets t
WHERE t.paid = FALSE
GROUP BY t.VIN;
```

b)
```
SELECT v.VIN, v.OwnerLicenseID
FROM Vehicles v, UnPaidTicketFees u1
WHERE v.VIN = u1.VIN
AND u1.UnPaidTotal =
  (SELECT MAX(u2.UnpaidTotal)
     FROM UnpaidTicketFees u2);
```