CSE 180, Final Exam, Fall 2019, Shel Finkelstein

Student Name: _____

Student ID: _____

UCSC Email: _____

Final Points

Part	Max Points	Points
I	42	
II	24	
III	36	
Total	102	

The first Section (Part I) of the Fall 2019 CSE 180 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 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. Also hand in your **8.5 x 11 "cheat sheet"**, with your name written in the upper right corner of the sheet.

Part II: (24 points, 6 points each)

Question 22: Assume that we already have:

- a *person* table, whose Primary Key is *SSN*, and
- a *house* table, whose Primary Key is *houseID*.

Write a Create statement for another table:

property(*owner, houseID, datePurchased*)

A *property* tuple indicates that a person (*owner*) purchased a house (*houseID*) on the specified datePurchased. *owner* and *SSN* are integers, and *houseID* is an integer in both *house* and *property*; *datePurchased* is a date.

The Primary Key of *property* is (*owner, houseID*). In the *property* table that you create, *owner* should be a Foreign Key corresponding to the Primary Key of *person* (*SSN*), and *houseID* should be a Foreign Key corresponding to the Primary Key of *house* (also called houseID).

Moreover, when a *person* is deleted, tuples in *property* which that person owns should be deleted. But deletion of a tuple in *house* is not permitted if there is a tuple in *property* for that *houseID*. (You don't have to consider updates in your Create.)

Answer 22:

```
CREATE TABLE property (
        owner INTEGER,
        houseID INTEGER,
        datePurchased DATE,
        PRIMARY KEY (owner, houseID),
        FOREIGN KEY (owner) REFERENCES person(SSN) ON DELETE CASCADE,
        );
```

The second FOREIGN KEY line could just read:

```
        FOREIGN KEY (houseID) REFERENCES house
```

Another correct answer:

```
CREATE TABLE property (
        owner INTEGER REFERENCES person(SSN) ON DELETE CASCADE,
        houseID INTEGER REFERENCES house(houseID),
        datePurchased DATE,
        PRIMARY KEY (owner, houseID)
        );
```

Question 23:

23a): Explain the meaning of Physical Independence for Relational Database.

Answer 23a):

Physical independence means that the actual physical representation of the data in tables doesn't affect the SQL statements that can be executed, except for performance. For example, data might be stored in a row-oriented manner or in a column-oriented manner, and queries don't have to be changed to reflect this. Moreover, queries don't have to be changed when access structures such as indexes are created or dropped.

23b): Why are Relational Database queries described as Declarative?

Answer 23b):

Relational database queries, which are written in SQL, describe what should be in the answers; they don't describe an execution plan for computing the answers. So queries are described as Declarative (or Non-Procedural) because the database management system is responsible for figuring out an execution plan that computes a query response that corresponds to the description in a SQL query.

23c): What's the connection between Physical Independence and Declarative queries for Relational Database?

Answer 23c):

The connection between Physical Independence and Declarative queries for Relational database is that the database management system (and in particular, the optimizer) figures out an execution plan based on the physical structure of the data that does what the Declarative query describes. The optimizer tries to choose an execution plan that is not only correct, but also has good performance.

Question 24: We have the following relations:

Sailors(sid, sname, rating, age) // sailor id, sailor name, rating, age
Boats(bid, bname, color) // boat id, boat name, color of boat
Reserves(sid, bid, day) // sailor id, boat id, date that sailor sid reserved boat bid.

Codd's relational algebra for sets included only the 5 operators Selection(σ), Projection(π), Product(x), Union ($\cup$) and Difference (-). Using only those 5 operators, write a Relational Algebra query that finds the names of sailors whose age is more than 20, but who have not reserved any boat whose color is red.

If you'd like, you may also use Rename (ρ) and Assignment ($\leftarrow$) in your answer.

Very Big Hint: In SQL, you can use NOT EXIST or NOT IN or COUNT, but you can't use any of those in Relational Algebra.

But you can use Difference. So first, find the names of sailors whose age is more than 20 and who have reserved a boat whose color is red. Then use Difference to find the answer.

Answer 24:

Here's one correct answer. Other correct answers might push selection and projection closer to the relations.

$$\pi_{\text{Sailors.sname}} (\sigma_{\text{Sailors.age} > 20} (\text{Sailors})) -$$

$$(\pi_{\text{Sailors.sname}} (\sigma_{\text{Sailors.age} > 20 \text{ AND Boats.color} = \text{'red'} \text{ AND Sailors.sid} = \text{Reserves.sid AND Boats.bid} = \text{Reserves.bid}} (\text{Sailors} \times \text{Reserves} \times \text{Boats})))$$

Question 25: Suppose that you have a relation:

BookInfo(Author, Citizenship, Title, PageCount)

with the following Functional Dependencies:

Author → Citizenship

Author, Title → PageCount

25a) Is {Author, Title} a key for BookInfo? Justify your answer fully and clearly.

Answer 25a):

Yes, {Author, Title} is a key for BookInfo.

Using the attribute closure algorithm, $\{Author, Title\}^+$ = {Author, Title, Citizenship, PageCount), which is all the attributes of BookInfo. That shows that {Author, Title} is a superkey for BookInfo.

To show that {Author, Title} is a key for BookInfo, we have to check to see if we'd still have a superkey if we eliminated an attribute from {Author, Title}.

$\{Author\}^+$ is {Author, Citizenship}, and that's not all the attributes of BookInfo.
$\{Title\}^+$ is {Title}, }, and that's not all the attributes of BookInfo.

So since {Author, Title} is a superkey which won't still be a key if it's "put on a diet", then {Author, Title} is a key for Bookinfo.

25b) Is BookInfo in BCNF? Justify your answer fully and clearly.

Answer 25b):

To be in BCNF, all the Functional Dependencies given for the BookInfo would have to either be trivial, or have left-hand sides that are superkeys. If we can find even one FD that is not trivial and also does not have a left-hand side that's a superkey, then BookInfo is not in BCNF.

Consider the FD: Author → Citizenship.

- This isn't a trivial FD, since Citizenship is not one of attributes on the left-hand side of the FD.
- Is the left-hand side of this FD a superkey? No, since $\{Author\}^+$ is just {Author, Citizenship}, and that's not all the attributes of BookInfo.

Hence BookInfo with these FDs is not in BCNF.

25c) Is BookInfo in 3NF? Justify your answer fully and clearly.
Answer 25c):

To be in 3NF, all the Functional Dependencies given for the BookInfo would have to either

a) be trivial, or
b) have left-hand sides that are superkeys, or
c) have a right-hand side that is part of some key.

If we can find even one FD that is none of these 3 things, then BookInfo is not in 3NF.

Consider the FD: Author → Citizenship. As we've already seen in 25b)

- This isn't a trivial FD, since Citizenship is not one of attributes on the left-hand side of the FD.
- Is the left-hand side of this FD a superkey? No, since $\{Author\}^+$ is just {Author, Citizenship}, and that's not all the attributes of BookInfo.

Well, now let's see whether the right-hand side of this FD, which is Citizenship, is part of some key for BookInfo. Suppose that we have a set of attributes S that's a superkey for BookInfo.

- Nothing determines Author except Author, so Author must be in S.
- Also, nothing determines Title except Title, so Title must be in S.

So any superkey for BookInfo must include Author and Title. Can there be a key that includes Citizenship? Well, such a key must include {Author, Title, Citizenship}, and it is true that {Author, Title, Citizenship} is a superkey, since it's closure is all the attributes of BookInfo. But {Author, Title, Citizenship} is not a key, since we can eliminate Citizenship an still have a superkey, {Author, Title.} Therefore there cannot be a key that includes Citizenship.

Hence BookInfo with these FDs is not in 3NF.

Part III: (36 points, 9 points each)

Some familiar tables appear below, with Primary Keys underlined. **These tables also appear on the last page of the Final, which you can tear off to help you do questions in Part III of the Final.** You don't have to turn in that last page at the end of the Exam.

customer(custID, name, address, joinDate, status)

menuItem(menuItemID, name, description, price)

dinnerTable(dinnerTableID, numSeats, InUse)

server(serverID, name, level, salary)

visit(visitID, custID, dinnerTableID, serverID, numPeople, cost, custArrive, custDepart)

billEntry(visitID, menuItemID, quantity)

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

You may assume Referential Integrity as follows:

- Each custID in visit appears as a Primary Key in customer.
- Each dinnerTableID in visit appears as a Primary Key in dinnerTable.
- Each serverID in visit appears as a Primary Key in server.
- Each visitID in billEntry appears as a Primary Key in visit.
- Each menuItemID in billEntry appears as a Primary Key in menuItem.

Write legal SQL queries for Questions 26-29. If you want to create and then use views to answer these questions, that's okay, but views are not required unless the question asks for them.

Don't use DISTINCT in your queries unless it's necessary, 1 point will be deducted if you use DISTINCT when you don't have to do so. And some points may be deducted for queries that are very complicated, even if they are correct.

Question 26: Find the menuItemID, name and price for every menuItem that has the highest price of any menuItem. Note that there may be more than one menuItem that has that highest price. Your result should appear in reverse alphabetical order based on name. No duplicates should appear in your result

Answer 26: Here are 3 correct solutions.

Solution #1:

```
SELECT m.menuItemID, m.name, m.price
FROM menuItem m
WHERE m.price >= ALL ( SELECT m2.price
                       FROM menuItem m2 )
ORDER BY m.name DESC;
```

Solution #2:

```
SELECT m.menuItemID, m.name, m.price
FROM menuItem m
WHERE NOT EXISTS ( SELECT m2.price
                   FROM menuItem m2
                   WHERE m2.price > m.price )
ORDER BY m.name DESC;
```

Solution #3:

```
SELECT m1.menuItemID, m1.name, m1.price
FROM menuItem m1
WHERE m1.price = ( SELECT MAX(price)
                   FROM menuItem )
ORDER BY m1.name DESC;
```

Question 27: Find the ID and name of each server whose level is 'B' and who was the server for a visit in which 'Apple Pie' was one of the items on the bill. No duplicates should appear in your result.

Answer 27: Here are 2 correct solutions. DISTINCT is needed in the first solution,, but not in the second.

Solution #1:

```
SELECT DISTINCT s.serverID, s.name
FROM server s, billentry b, visit v, menuitem m
WHERE s.level = 'B'
      AND s.serverid = v.serverid
      AND v.visitid = b.visitid
      AND m.menuitemid = b.menuitemID
      AND m.name = 'Apple Pie';
```

Solution #2:

```
SELECT s.serverID, s.name
FROM server s
WHERE s.level = 'B'
    AND s.serverID IN ( SELECT v.serverID
                        FROM visit v, billEntry b, menuItem m
                        WHERE v.visitID = b.visitID
                            AND b.menuItemID = m.menuItemID
                            AND m.name = 'Apple Pie'
                      );
```

Question 28: For each visit, there may be billEntry tuples for that visit. Let's define a *billedVisit* to be a visit for which there is at least one billEntry tuple.

Each billEntry tuple has a quantity, and each billEntry tuple has a menuItemID that identifies a menuItem tuple that has a price. So the calculatedCost of a *billedVisit* can be computed by adding up price*quantity for all of the billEntry tuples for that *billedVisit*.

The visit table has a cost attribute. Find all the *billedVisits* for which the cost of that visit is not equal to the calculated cost of that visit. In your result, the attributes should be visitID, cost and calculatedCost. No duplicates should appear in your result.

Answer 28: Here are 2 solutions, one without a view, and the other with a view.

Solution #1:

```
SELECT b.visitID AS visitID, v.cost AS cost,
                SUM(b.quantity * m.price) AS calculatedCost
FROM visit v, billentry b, menuitem m
WHERE v.visitID = b.visitID
    AND b.menuitemid = m.menuitemid
GROUP BY b.visitid, v.cost
HAVING SUM(b.quantity * m.price) <> v.cost;
```

Solution #2:

```
CREATE VIEW visitBill AS
        SELECT b.visitID, SUM(m.price* b.quantity) AS calculatedCost
        FROM billEntry b, menuItem m
        WHERE m.menuitemID = b.menuitemID
        GROUP BY b.visitID;

SELECT v.visitID, v.cost, vb.calculatedCost
FROM visit v, visitBill vb
WHERE v.visitID = vb.visitID
    AND v.cost <> vb.calculatedCost;
```

Question 29: This question has two parts; be sure to answer both.

29a): Create a SQL view called custVisitCount. For each customer who joined before December 25, 2018, the custVisitCount view should give the custID, address and the number of visits for that customer. In your view, the attributes should be called theCustomer, theAddress and theVisitCount.

[Hint: There could be customers who have no visits, and theVisitCount for them should be 0. If your view works correctly just for customers who do have visits, you will only lose 1 point for that.]

29b): Use the view custVisitCount to find the address of each customer such that:

- the customer visited 8 or more times, and also
- no other customer in the customer table has the same address.

[Yes, you'll also have to use the customer table to do this.]

The only attribute in your result should be theAddress. No duplicates should appear in your result.

Answer 29a): Here are two solutions.

Solution #1:

```
CREATE VIEW custVisitCount AS
        SELECT c.custID as theCustomer, c.address AS theAddress,
                COUNT(v.visitid) AS theVisitCount
        FROM customer c, visit v
        WHERE c.joinDate < DATE '2018-12-25'
            AND c.custID = v.custID
        GROUP BY c.custID, c.address

        UNION

        SELECT c.custID as theCustomer, c.address AS theAddress,
                0 AS theVisitCount
        FROM customer c
        WHERE c.joinDate < DATE '2018-12-25'
            AND NOT EXISTS ( SELECT *
                             FROM visits
                             WHERE c.custID = v.custID );
```

29a) Solution #2:

```
CREATE VIEW custVisitCount AS
        SELECT c.custID AS theCustomer, c.address AS theAddress,
                COUNT(v.visitID) AS theVisitCount
        FROM customer c LEFT OUTER JOIN visit v
                ON c.custID = v.custID
        WHERE c.joinDate > DATE '2018-12-25'
        GROUP BY c.custID, c.address;
```

Answer 29b): Here are two solutions; there are others.

Solution #1:

```
SELECT cvc.theAddress
FROM custVisitCount cvc
WHERE cvc.theVisitCount >= 8
   AND NOT EXISTS ( SELECT c.address
                    FROM customer c
                    WHERE c.address = cvc.theAddress );
```

Solution #2:

```
SELECT cvc.theAddress
FROM custVisitCount cvc
WHERE cvc.thevisitCount >= 8
     AND cvc.theaddress IN ( SELECT c.address
                             FROM Customer c
                             GROUP BY c.address
                             HAVING COUNT(c.address) = 1 );
```

This is extra blank space, in case you need more paper to answer questions. Write Question number clearly if you use blank pages.

You may tear off this page to help you do Part III of the Final.

Some familiar tables appear below, with Primary Keys underlined. **These tables also appear on the last page of the Final, which you can tear off to help you do questions in Part III of the Final.** You don't have to turn in that last page at the end of the Exam.

customer(custID, name, address, joinDate, status)

menuItem(menuItemID, name, description, price)

dinnerTable(dinnerTableID, numSeats, InUse)

server(serverID, name, level, salary)

visit(visitID, custID, dinnerTableID, serverID, numPeople, cost, custArrive, custDepart)

billEntry(visitID, menuItemID, quantity)

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

You may assume Referential Integrity as follows:

- Each custID in visit appears as a Primary Key in customer.
- Each dinnerTableID in visit appears as a Primary Key in dinnerTable.
- Each serverID in visit appears as a Primary Key in server.
- Each visitID in billEntry appears as a Primary Key in visit.
- Each menuItemID in billEntry appears as a Primary Key in menuItem.

Write legal SQL queries for Questions 26-29. If you want to create and then use views to answer these questions, that's okay, but views are not required unless the question asks for them.

Don't use DISTINCT in your queries unless it's necessary, 1 point will be deducted if you use DISTINCT when you don't have to do so. And some points may be deducted for queries that are very complicated, even if they are correct.