CMPS 180, Final Exam, Fall 2017, Shel Finkelstein

Student Name: _____

Student ID: _____

UCSC Email: _____

Final Points

Part	Max Points	Points
I	40	
II	30	
III	36	
Total	106	

The first Section (Part I) of the Winter 2017 CMPS 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: (30 points, 6 points each)

Question 21: Assume that we already have a Persons table, whose Primary Key is SSN, and a Houses table, whose Primary Key is HouseID.

Write a Create statement for another table:
Properties(Owner, HouseID, DatePurchased)
which describes houses owned by persons. Owner and SSN are integers, and HouseID is an integer in both Houses and Properties; DatePurchased is a date.

The Primary Key of Properties is (Owner, HouseID). In the Properties table that you create, Owner should be a Foreign Key corresponding to the Primary Key of Persons (SSN), and HouseID should be a Foreign Key corresponding to the Primary Key of Houses (which is also called HouseID).

Moreover, when an Owner is deleted, tuples in Properties for that Owner should be deleted. But deletion of a tuple in Houses is not permitted if there is a tuple in Properties for that HouseID. (You don't have to consider updates in your Create.)

Answer 21:

```
CREATE TABLE Properties (
  Owner INTEGER REFERENCES Persons (SSN) ON DELETE CASCADE,
  HouseID INTEGER REFERENCES Houses (HouseID),
  DatePurchased DATE,
  PRIMARY KEY (Owner, HouseID)
);
```

or

```
CREATE TABLE Properties (
  Owner INTEGER,
  HouseID INTEGER,
  DatePurchased DATE,
  PRIMARY KEY (Owner, HouseID),
  FOREIGN KEY (Owner) REFERENCES Persons (SSN) ON DELETE CASCADE,
  FOREIGN KEY (HouseID) REFERENCES Houses (HouseID)
);
```

Question 22: Assume that Employees(empid, name, dept, salary) is table that has a large number of tuples, with many employees who are in many different departments (dept). For the following query and indexes:

```
SELECT name
FROM Employees
WHERE dept =' Engineering' and salary > 5000;
```

```
CREATE INDEX IX_dept ON Employees(dept);
CREATE INDEX IX_salary ON Employees(salary);
CREATE INDEX IX_name ON Employees(name);
CREATE INDEX IX _dept_salary ON Employees(dept, salary);
```

22a): Give each of the four indexes one of the following evaluations:

- NotGood (unlikely to help process the query efficiently)
- Good (likely to help process the query efficiently)
- Best (only one index can get this evaluation)

Answer 22a):

IX_dept Evaluation: ____Good_____

IX_salary Evaluation: ____Good_____

IX_name Evaluation: ____NotGood_____

IX _dept_salary Evaluation: ____Best_____

22b): Suppose that those 4 indexes all existed, and there was an UPDATE statement that transferred someone named Edward Lee (who was already in Employees) to the Marketing department. Would any of these 4 indexes have to be updated? If so, which one(s)?

Answer 22b):

Yes, just IX_dept and IX_dept_salary

Question 23: 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 blue.

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

Answer 23:

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

$$\Pi_{\text{Sailors.sname}} (\sigma_{\text{Sailors.age} > 20} (\text{Sailors})) \; - \; (\Pi_{\text{Sailors.sname}} (\sigma_{\text{Sailors.age} > 20 \text{ AND Boats.color} = \text{'Blue' AND Sailors.sid} = \text{Reserves.sid AND Boats.bid} = \text{Reserves.bid}} (\text{Sailors} \times \text{Reserves} \times \text{Boats})))$$

Question 24: Suppose that you have a relation

CompanyData(Department, Project, Manager)

with the following non-trivial functional dependencies:

Department, Project → Manager

Manager → Project

24a) Is CompanyData in BCNF? Justify why your answer is correct fully and clearly.
Answer 24a):

No, CompanyData is not in BCNF. To be in BCNF, each FD would have either be trivial, or have a left-hand side that is a superkey. Trivial FDs are always true, but we don't bother to specify them.

The second FD is not trivial. Also, for that second FD, the left-hand side, Manager, is not a superkey, since Manager$^+$ is {Manager, Project}, which is not all the attributes of CompanyData.

24b): Is CompanyData in 3NF? Justify why your answer is correct fully and clearly.
Answer 24b):

Yes, CompanyData is in 3NF. To be in 3NF, each FD would have either be trivial, or have a left-hand side that is a superkey, or have a right-hand side that is part of a superkey. Trivial FDs are always true, but we don't bother to specify them.

For the first FD, the left-hand side is a superkey, since {Department, Project}$^+$ is {Department, Project, Manager}, which is all the attributes of CompanyData.

For the second FD, the right-hand side (Project) is part of a key, namely (Department, Project). We know that (Department,Project) is a key because its attribute closure is all the attributes of CompanyData (as we just saw). But Department$^+$ is just Department, while Project$^+$ is {Project,Manager}. Taking an attribute out of the superkey (Department,Project) gives us a result that is <u>not</u> a superkey, so (Department,Project) is a key.

Since all of the FDs satisfy the criteria, CompanyData is in 3NF.

Question 25: The Tickets table, whose Primary Key is TicketID, is as follows:

Tickets (TicketID, CustID, AirlineID, FlightNum, SeatNum, Cost, Paid)

Here's part of a PostgreSQL plpsql Stored Function called CostCut. The Stored Function has a float parameter (reduction_limit), and it returns a float value, described below. CostCut should cut the price of some Tickets in half, so (for example) a ticket that cost 500 would instead cost 250, a reduction of 250.

But CostCut shouldn't reduce the price of all tickets. It should halve the price of the tickets with the highest cost, keeping track of the total reduction of cost as it goes. When the total reduction becomes ≥ reduction_limit, CostCut shouldn't do any more reductions. It should just return the total reduction of cost that it has computed.

```
CREATE FUNCTION costCut (reduction_limit float)
RETURNS float AS
$$

   // Declarations:  Fill in the declarations for any local variables and cursors

BEGIN
    // Body:  Fill in the body of the code for CostCut
   // Remember that you can use found to see if cursor found anything
END;
$$
LANGUAGE plpgsql
```

Answer 25: Write the Declarations and the Body below, clearly saying which is which. You don't have to repeat the parts of the Stored Function that appear above.

Declarations:

```
DECLARE cursor_c AS cursor FOR
        SELECT cost FROM tickets ORDER BY cost DESC FOR UPDATE;
        DECLARE total FLOAT;
        DECLARE fetched_cost FLOAT;
```

Body:

```
OPEN cursor_c;
total := 0;
FETCH cursor_c INTO fetched_cost;
WHILE total < reduction_limit AND found
LOOP
        UPDATE tickets SET cost = fetched_cost/2 WHERE CURRENT OF cursor_c
        total := total + fetched_cost/2;
        FETCH cursor_c INTO fetched_cost;
END LOOP;
CLOSE cursor_c;    // Not required, but okay
```

RETURN total;

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.

Airlines (AirlineID, AirlineName)
Airports (AirportID, City, State)
Flights (AirlineID, FlightNum, Origin, Destination, DepartureTime, ArrivalTime)
Customers (CustID, CustName, Status)
Tickets (TicketID, CustID, AirlineID, FlightNum, SeatNum, Cost, Paid)

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

You may assume Referential Integrity as follows:
- Each CustID in Tickets appears as a Primary Key in Customers.
- Each (AirlineID, FlightNum) in Tickets appears as a Primary Key in Flights.
- Each Origin and each Destination in Flights appears as a Primary Key in Airports.
- Each AirlineID in Flights appears as a Primary Key in Airlines.

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, 0.5 points 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: There might be airports for which no flights go to that airport; that is, airports where no flight's destination is that airport. Find the AirportID, City and State of each airport for which no flights go to that airport. The results of your query should appear in reverse alphabetical order by State, and for each State, in reverse alphabetical order by City.

Answer 26:

Here are 3 correct answers.

```
SELECT a.AirportID, a.City, a.State
FROM Airports a
WHERE NOT EXISTS (
        SELECT *
        FROM Flights f
        WHERE f.Destination = a.airportID )
ORDER BY a.State DESC, a.City DESC;
```

```
SELECT a.AirportID, a.City, a.State
FROM Airports a
WHERE a.airportID NOT IN (
                SELECT f. Destination
                FROM Flights f )
ORDER BY a.State DESC, a.City DESC;
```

```
SELECT a.AirportID, a.City, a.State
FROM Airports a
WHERE a.airport <> ALL (
        SELECT Destination
        FROM Flights f )
ORDER BY a.State DESC, a.City DESC;
```

Question 27: For each ticket that appears in Tickets for which the customer on that ticket has Status 'B', your SQL query should output the TicketID, the name of the customer who has that ticket, the State for the Origin of the flight on the ticket, and the State for the Destination of the flight on the ticket. The attribute names that appear in your result are up to you, but they should all be different.

Answer 27:

```
SELECT t.TicketID, c.Custname, a1.State AS origin_state,
                                    a2.State AS destination_state
FROM Tickets t, Customers c, Flights f, Airports a1, Airports a2
WHERE c.Custid = t.Custid
   AND c.Status = 'B'
   AND f.FlightNum = t.FlightNum
   AND f.Origin = a1.AirportID
   AND f.Destination = a2.AirportID;
```

DISTINCT is not needed for this query, since TicketID is the primary key for Tickets, and at most one result can appear for each TicketID.

Question 28: We're looking for flights that go from airport EWR to airport SFO with exactly one stopover airport (so there's a first flight from EWR to some stopover airport, and a second flight from the same stopover airport to SFO). The time spent at the stopover airport should be at least one hour, but should also be no more than 90 minutes.

Your result should include FlightNum1, DepartureTime1, ArrivalTime1, StopOverAirportID, FlightNum2, DepartureTime2, ArrivalTime2. Use those names for the attributes in your result.

Answer 28:

```
SELECT f1.FlightNum AS FlightNum1, f1.DepartureTime AS DepartureTime1,
    f1.ArrivalTime AS ArrivalTime1, f1.Destination AS StopOverAirportID,
   f2.FlightNum AS FlightNum2, f2.DepartureTime AS DepartureTime2,
   f2.ArrivalTime AS ArrivalTime2
FROM flights f1, flights f2
WHERE f1.Origin = 'EWR'
   AND f2.Destination = 'SFO'
   AND f1.Destination = f2.Origin
   AND f2.DepartureTime - f1.ArrivalTime >= INTERVAL '1 HOUR'
   AND f2.DepartureTime - f1.ArrivalTime <= INTERVAL '90 MINUTES';
```

No error whether or not you use DISTINCT, since problem was ambiguous.

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

29a): Create a SQL view called NonStops. For each pair of airports, the NonStops view should give the first airport, the second airport, and the number of flights whose origin is the first airport and whose destination is the second airport. In your view, the attributes should be called Origin, Destination and FlightCount.

[Hint: Doing this correctly may be difficult, since there could be pairs of Airports that don't have flights between them. If your view works correctly just for Airports that do have flights between them, you will only lose 1 point for that.]

29b): Using the view NonStops (not the original tables), find the total number of flights that go to each Destination. Your result should have attributes Destination and TotalFlights. But only include a Destination in your result if there are 3 or more tuples in NonStops which have that Destination value.

Answers 29a) and 29b):

Answer 29a):

The following view gives the result without providing count for Airports that have no flights between them. You would lose only 1 point for this solution.

```
CREATE VIEW NonStops AS
   ( SELECT f.Origin, f.Destination, COUNT(*) AS FlightCount
     FROM Flights f
     GROUP BY f.Origin, f.Destination );
```

The following view gives the result and also provides 0 as the count for Airports that have no flights between them. A second way to do this (using Outer Join) follows on the next page.

```
CREATE VIEW NonStops AS
   ( SELECT f.Origin, f.Destination, COUNT(*) AS FlightCount
     FROM Flights f
     GROUP BY f.Origin, f.Destination )
UNION
   (  SELECT a1.airportid, a2.airportid, 0 AS FlightCount
     FROM Airports a1, Airports a2
     WHERE NOT EXISTS
          ( SELECT * FROM Flights f
            WHERE f.Origin = a1.AirportID)
               AND f.Destination = a2.AirportID)  );
```

Here's a view that uses Outer Join instead, also providing 0 as the count for Airports that have no flights between them. To make this relatively easy to undersand, we'll create another view first that includes all pairs of airports (no WHERE clause, so it's a Cartesian product). Then we take the Left Outer Join of that with Flights. This seems clearer than writing a query that has both Join and Left Outer Join in it (which also would work).

```
CREATE VIEW AirportPairs AS
   ( SELECT a1.AirportID AS firstAirport, a2.AirportID AS secondAirport
     FROM Airports a1, Airports a2 );

CREATE VIEW NonStops AS
   ( SELECT ap.firstAirport, ap.secondAirport, COUNT(f.FlightID) AS FlightCount
     FROM AirportPairs ap LEFT OUTER JOIN Flights f
     WHERE ap.firstAirport = f.Origin
          AND ap.secondAirport = f.Destination
     GROUP BY ap.firstAirport, ap.secondAirport );
```

For an AirportPair that has no flights between the airports, there will be a single tuple in the Left Outer Join that has NULL as its f.FlightID value. For such a pair of airports, COUNT(f.FlightID) will be 0 in NonStops. So by using Left Outer Join (much the way it can be used for OLAP), we avoided need for Union in previous definition of the NonStops view.

Answer 29b):

```
SELECT n.Destination, SUM(n.FlightCount) AS TotalFlights
FROM NonStops n
GROUP BY n.Destination
HAVING COUNT(*) >= 3 ;
```

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.

Airlines (AirlineID, AirlineName)
Airports (AirportID, City, State)
Flights (AirlineID, FlightNum, Origin, Destination, DepartureTime, ArrivalTime)
Customers (CustID, CustName, Status)
Tickets (TicketID, CustID, AirlineID, FlightNum, SeatNum, Cost, Paid)

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

You may assume Referential Integrity as follows:

- Each CustID in Tickets appears as a Primary Key in Customers.
- Each (AirlineID, FlightNum) in Tickets appears as a Primary Key in Flights.
- Each Origin and each Destination in Flights appears as a Primary Key in Airports.
- Each AirlineID in Flights appears as a Primary Key in Airlines.

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, 0.5 points will be deducted