CMPS 180, Midterm Exam, Winter 2016, Shel Finkelstein

Student Name: _____

Student ID: _____

UCSC Email: _____

Midterm Points

Part	Max Points	Points
I	24	
II	30	
III	18	
IV	30	
Total	102	

If you need additional space, you may answer problems on the blank pages of the exam, but be sure to number problems when you do that.

Please show your **UCSC ID** when you turn in your exam.

Part I: (24 points, 4 each)

The following **TRUE** or **FALSE** questions refer to the Customers(cid, cname, type, level, age) relation, where cid is the primary key. cid and age are INTEGER and cname, type and level are CHAR(20).

Question 1: TRUE or **FALSE:** If Customers has 7 tuples in it, then there will be 49 tuples in the result of the query:

```
SELECT *
FROM Customers, Customers;
```

Answer 1: ___ TRUE _____

Question 2: TRUE or **FALSE** : The two queries below are equivalent, meaning that they always have the same answer, no matter what's in the Customers relation.

```
SELECT c.cname
FROM Customers c
WHERE c.age > ALL  (SELECT c2.age
                    FROM Customers c2
                    WHERE c2.type = 'snowboard');
```

```
SELECT c.cname
FROM Customers c
WHERE c.age > (SELECT MAX(c2.age)
               FROM Customers c2
               WHERE c2.type = 'snowboard');
```

Answer 2: ___ FALSE _____

Question 3: TRUE or **FALSE:** The following is a legal SQL query.

```
SELECT level,  MAX(age), type
FROM Customer
WHERE type <> 'snowboard'
GROUP BY level;
```

Answer 3: ___ FALSE _____

Question 4: If an instance of Customers has 1000 tuples in it, and the value of cname is different for every tuple in that instance, then the CREATE statement for Customers must have specified that cname is UNIQUE.

Answer 4: ___ FALSE _____

Question 5:

If we execute a transaction using Isolation Level READ COMMITTED, and if we execute the following statement twice during that transaction:

```
SELECT level
FROM Customers
WHERE cid=123;
```

then the value of level in the query's result must be the same both times that the statement is executed in the transaction.

Answer 5: ___ FALSE _____

Question 6:

If Customers has only one tuple, and the value of age is NULL for that tuple, then the result of the query:

```
SELECT SUM(age), COUNT(age)
FROM Customers;
```

will be a single tuple that has values (NULL, 0).

Answer 6: ___ TRUE _____

Part II: (30 points, 6 each):

Question 7: CREATE statements can specify attributes as PRIMARY KEY, and may also specify attributes as UNIQUE. Explain two differences between PRIMARY KEY and UNIQUE.

Answer 7a:

PRIMARY KEY attributes cannot have NULL values. On the other hand, attributes defined as UNIQUE can have NULL values.

Answer 7b:

A table can only have one primary key constraint, but can have multiple unique constraints.

Question 8: If Students is a relation in your database, what would each of the following two statements do? Are their effects different?

DELETE FROM Students;

DROP TABLE Students;

Answer 8:

Yes, they are different.

The DELETE statement above deletes all the tuples from the table, but the table still exists. For example, rows can be inserted into the table after the DELETE.

The DROP statement removes the table from the database completely. For example, rows cannot be inserted after the table after the DROP, because the TABLE no longer exists.

Question 9: Explain the meanings of UNION and UNION ALL in SQL. After explaining the meanings, answer the following question: If a tuple appears 12 times in the answer to Q1 and 3 times in the answer to Q2, how many times would it appear in Q1 UNION Q2, and how many times would it appear in Q1 UNION ALL Q2.

Answer 9:

Q1 UNION Q2; will return the union of the two queries, that is, the set of tuples that are present in the output of either Q1 or Q2, eliminating duplicates from the result.

Q1 UNION ALL Q2 will return the union without eliminating duplicates.

If a tuple appears 12 times in Q1 and 3 in Q2, it will appear only once in Q1 UNION Q2, and it will appear 15 times in Q1 UNION ALL Q2.

Question 10: Views can be used as if they were tables in most but not all ways. Provide clear examples:

a) where a view can be used as if it was a table and
b) where a view cannot be used as if it's a table.

Answer 10a:

A view can be used in the FROM clause of an SQL query in the same way as a table. It can be queried, joined, and even other views can be defined on the view.

Answer 10b:

Modification statements that would work on Tables may result in errors when applied to Views.

For example, an INSERT into a View that has a subset of the columns in the underlying Table will result in an error if values for the remaining columns don't have DEFAULT values and also cannot be NULL. (See Slide 18 of Lecture 7.)

Also, UPDATE of a View will result in an error if the UPDATE is not well-defined, even though UPDATE of a Table would work. For example, if we have tables:

Employees(ename, edept)
Departments(dname, dmanager)

with a view:

```
CREATE VIEW EmpManager(name, manager) AS
        SELECT e.ename, d.dmanager
        FROM Employees e, Departments d
        WHERE e.edept = d.dname;
```

then the statement that changes Smith's manager to Lee::

```
UPDATE EmpManager
SET manager = 'Lee'
WHERE name = 'Smith";
```

will fail because it's not clear what to do on the underlying Employees and Departments tables. Does Smith get moved to a department that Lee manages, or does Lee become the manager of the department that Smith is in?

Question 11: Provide the Truth Table for the condition "p AND q" in SQL's three-valued logic (TRUE, FALSE, UNKNOWN).

Answer 11:

p	p	p AND q
True	True	True
True	False	False
True	Unknown	Unknown
False	True	False
False	False	False
False	Unknown	False
Unknown	True	Unknown
Unknown	False	False
Unknown	Unknown	Unknown

Part III: (18 points, 6 each):

The multiple choice questions below are about the following relations. The primary key is underlined in each relation.

Beers(<u>name</u>, manufacturer) // For each beer, gives the beer's manufacturer
Sells(<u>bar, beer</u>, price) // For each bar and beer, gives beer's price at bar
Likes(<u>drinker, beer</u>) // Indicates that drinker likes a beer

What does each SQL statement do? For each question, give the best answer.

Question 12:

```
SELECT b.name
FROM Beers b, Likes lk1, Likes lk2
WHERE b.name=lk1.beer AND b.name=lk2.beer AND lk1.drinker <> lk2.drinker;
```

a) Finds names of all the different beers that at least one drinker likes.
b) Finds names of all the different beers that at least two drinkers like.
c) Finds names of all beers that at least two drinkers like, with duplicates possible.
d) Finds all drinkers that like at least two different beers, with duplicates possible.
e) None of the above

Answer 12: ____c_____

Duplicates for the beer name will appear in the answer once for each pair of folks who like the beer. Note that even if there are only two people Smith and Jones who like a beer, there will be a duplicate because the beer will appear for lk1 Smith and lk2 Jones, and it will appear for lk1 Jones and lk2 Smith.

Question 13:

```
SELECT b.name
FROM Beers b
WHERE EXISTS
    ( SELECT *
      FROM Likes lk1, Likes lk2
       WHERE lk1.beer <> lk2.beer );
```

a) Finds all the different beers that at least one drinker likes.
b) Finds all the different beers that at least two drinkers like.
c) Finds all the beers that at least two drinkers like, with duplicates possible.
d) Finds all drinkers that like at least two different beers, with duplicates possible.
e) None of the above

Answer 13: ____e_____

The answer is e because the conditions b.name = lk1.beer AND b.name = lk2.beer are missing in subquery. Compare this to query in Question 12.

Question 14:

```
SELECT s1.bar
FROM Sells s1
WHERE s1.price < ANY ( SELECT s2.price
                       FROM Sells s2
                       WHERE s2.bar = 'Cheers' );
```

a) Finds bars that sell a beer that has a lower price than all the beers sold at the bar 'Cheers', with no duplicates.
b) Finds bars that sell a beer that has a lower price than all the beers sold at the bar 'Cheers', with duplicates possible.
c) Finds bars that sell a beer that has a lower price than at least one beer sold at the bar 'Cheers', with no duplicates.
d) Finds bars that sell a beer that has a lower price than at least one beer sold at the bar 'Cheers', with duplicates possible.
e) None of the above

Answer 14: ___d_____

Part IV: (30 points, 10 each):

The familiar relations Persons, Houses and Tenants are shown below, with Primary Keys underlined. Assume that all attribute values are NOT NULL. **Also assume that HouseAddress is UNIQUE in the Houses table.**

Persons (SSN, Name, HouseID, ApartmentNumber, Salary)

Houses (HouseID, HouseAddress, ApartmentCount, Color)

Tenants (HouseID, ApartmentNumber, LeaseTenantSSN, LeaseStartDate, LeaseExpirationDate, Rent, LastRentPaidDate, RentOverdue)

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 it's not required.

Question 15: Find the SSN and name for each person whose salary is more than 50000.

Answer 15:

```
SELECT SSN, Name
FROM Persons
WHERE Salary > 50000;
```

Question 16: Find the name of every person who lives in ApartmentNumber 8 of the House whose address is '650 Main Street'. The names in your result should appear in alphabetical order, as in a dictionary. No name should appear more than once in your result.

Answer 16:

Here are some of the ways to answer this question correctly; there are others. ASC is the default, so including it in the ORDER BY isn't needed but is okay.

```
SELECT DISTINCT P.Name
FROM Persons P, Houses H
WHERE P.ApartmentNumber = 8 AND P.HouseID = H.HouseID
   AND H.HouseAddress = '650 Main Street'
ORDER BY P.Name ASC;
```

```
SELECT DISTINCT P.Name
FROM Persons P
WHERE P.ApartmentNumber = 8 AND P.HouseID IN
        (SELECT H.HouseID From Houses H
        WHERE H.HouseAddress = '650 Main Street')
ORDER BY P.Name ASC;
```

```
SELECT DISTINCT P.Name
FROM Persons P
WHERE P.ApartmentNumber = 8 AND P.HouseID = ANY
        (SELECT H.HouseID From Houses H
        WHERE H.HouseAddress = '650 Main Street')
ORDER BY P.Name ASC;
```

Since we assume that HouseAddress is UNIQUE in Houses, the following are also correct, although the first may be error-prone and the second is ugly.

```
SELECT DISTINCT P.Name
FROM Persons P
WHERE P.ApartmentNumber = 8 AND P.HouseID =
        (SELECT H.HouseID From Houses H
        WHERE H.HouseAddress = '650 Main Street')
ORDER BY P.Name ASC;
```

```
SELECT DISTINCT P.Name
FROM Persons P
WHERE P.ApartmentNumber = 8 AND P.HouseID = ALL
        (SELECT H.HouseID From Houses H
        WHERE H.HouseAddress = '650 Main Street')
ORDER BY P.Name ASC;
```

Question 17: ApartmentCount tells us how many apartments there are in a house, but some apartments may not have tenants leasing them. The Tenants relation tells us which apartments have lease tenants.

For each house, find the number of apartments in that house that have lease tenants, the LeaseTenantCount. Your result should show the HouseID and LeaseTenantCount for each house. But only include a house in your result if that house has more than 5 lease tenants.

In your result, call the second attribute LeaseTenantCount.

Answer 17:

```
SELECT T.HouseID, COUNT(*) AS LeaseTenantCount
FROM Tenants T
GROUP BY T.HouseID
HAVING COUNT(*) >  5;
```

The COUNT(*) aggregate could be COUNT(T.LeaseTenantSSN) instead. The tuple variable T is not needed in this query.

If you chose to confirm that that a HouseID that appears in Tenants also appears in Houses, that's okay, as long as you write the query correctly. HouseID is the primary key for Houses, so there can only be one match. So the following solutions are also correct.

```
SELECT T.HouseID, COUNT(*) AS LeaseTenantCount
FROM Tenants T, Houses H
WHERE T.HouseID = H.HouseID
GROUP BY T.HouseID
HAVING COUNT(*) >  5;
```

```
SELECT T.HouseID, COUNT(*) AS LeaseTenantCount
FROM Tenants T,
WHERE EXISTS ( SELECT *
                    FROM Houses
                    WHERE T.HouseID = H.HouseID )
GROUP BY T.HouseID
HAVING COUNT(*) >  5;
```

Once again, the COUNT(*) aggregate could be COUNT(T.LeaseTenantSSN) instead. Tuples variables (or explicit mentions of the Tenants and Houses tables) are needed to that HouseID isn't ambiguous.