CSE 180, Midterm Exam, Winter 2020, Shel Finkelstein

Student Name: ______________________________

Student ID: ______________________________

UCSC Email: ______________________________

Midterm Points

Part	Max Points	Points
I	30	
II	24	
III	20	
IV	30	
Total	104	

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

This exam is single sided, and you may write an answer on the blank side of a page, if you need more space. But if you do that, clearly identify question that you've answered.

Note: For questions that ask you to write SQL statement, points might be deducted if you write complicated queries, even if they are correct. You can lose credit if you use DISTINCT but it's not needed, and you will lose credit if you don't use DISTINCT and it is needed.

Part I: (30 points, 6 each):

Question 1: R1, R2 and R3 are relation instances. If R1 has 5 tuples in it, R2 has 4 tuples in it, and R3 has 6 tuples in it. How many tuples are there in the result of the following SQL query?

```
SELECT *
FROM R1, R2, R3;
```

Answer 1: ____**120**____ 5*4*6 in Cartesian Product

Question 2: Let R(A,B,C,D) be a relation, where (A, B, C) is the primary key for R, and attribute D can be NULL. Assume that attribute A's domain has 6 different values, B's domain has 2 different values, C's domain has 3 different values, and D's domain has 5 different values. What is the maximum number of tuples that can be in an instance of R?

Answer 2: ____**36**_____ 6*2*3 due to Primary Key constraint

Question 3: NULL, UNKNOWN and the Empty Set have different meanings in SQL. Give an example of an expression that could have each of these values.

3a): NULL
Answer 3a):
An attribute of a table can be NULL (e.g., salary) if it's not part of Primary Key, and it doesn't have a NOT NULL constraint associated with it. Expressions involving such attributes (e.g., salary + 1000) can also be NULL.

3b): UNKNOWN
Answer 3b):
Any comparison predicate (other than IS NULL or IS NOT NULL) involving a NULL value can have the value UNKNOWN. For example, if salary has the value NULL, then "salary > 50000" will have the value UNKNOWN

3c): Empty Set
Answer 3c):
The result of a query (or subquery) is a set of tuples, which could be the Empty Set.

Question 4: If a tuple appears 20 times in the result for Q1 and that exact same tuple appears 8 times in the result for Q2, then:

4a): How many times would that tuple appear in Q1 INTERSECT Q2?

Answer 4a): _____**1**_____ min(20,8), with duplicates eliminated

4b): How many times would that tuple appear in Q1 INTERSECT ALL Q2?

Answer 4b): _____**8**_____ min(20,8)

4c): How many times would that tuple appear in Q1 UNION ALL Q2?

Answer 4c): _____**28**____ 20 + 8

Question 5: Your answers to the following questions should provide a brief explanation, not just some buzzwords. Give just one answer for each.

5a): Explain one advantage of views.
Answer 5a): Any one of the following answers is correct.

- Short-hand/encapsulation: You can treat a view as a table (for queries), which allows you to define a short-hand for a complicated expression.
- Re-use: You can re-use a view as often as you like; other people who can access the view may also re-use it.
- Authorization: People may be granted access to a view, even though they don't have access to the underlying tables.
- Logical data independence: Even if the tables underlying a view change, the view may still be used in queries, without re-writing the queries.

5b): Explain one advantage of indexes.
Answer 5b): Any one of the following answers is correct.

- Execution time of queries can be faster using indexes, since you can sometimes find the tuples matching a condition (such as "state = 'CA') faster using an index (on the state attribute), rather than scanning all the tuples in a table.
- If there is an index on the attributes of the Primary Key of a table, then the Primary Key constraint can be validated faster than if there wasn't an index on the Primary Key attributes. (Indexes also help with UNIQUE constraints.)

5c): Explain one disadvantage of indexes.
Answer 5c): Any one of the following answers is correct.

- There can be a huge (exponential) number of indexes on attribute combinations
- Indexes have to be stored, potentially taking up a lot of space.
- When indexes are searched, that uses memory/cache.
- Indexes have to be updated when tuples are inserted/deleted, as well as when there's an update of an attribute on which the index is defined .

Part II: (24 points, 8 each): The questions in Parts II and IV ask you to write SQL statements using the tables shown below, which are 3 of the tables in our Lab Assignments.

The primary key in each table is shown underlined. Assume that there aren't any NOT NULL or UNIQUE constraints specified for these tables. You may assume the same Referential Integrity constraints that we had in our Lab Assignment tables. Data types aren't shown to keep thing simple. There aren't any trick questions about data types.

Movies(<u>movieID</u>, name, year, rating, length, totalEarned)
Theaters(<u>theaterID</u>, address, numSeats)
Showings(<u>theaterID, showingDate, startTime</u>, movieID, priceCode)

Question 6: Write a SQL statement outputs the movieID values for Movies whose year is before 2011 and whose length isn't NULL. No duplicates should appear in your result.

Answer 6:

```
SELECT movieID
FROM Movies
WHERE year < 2011
    AND length IS NOT NULL;
```

Shouldn't write SELECT DISTINCT, since there can't be multiple movies with the same movieID. (Using DISTINCT would be correct, but unnecessary.)

Movies(<u>movieID</u>, name, year, rating, length, totalEarned)
Theaters(<u>theaterID</u>, address, numSeats)
Showings(<u>theaterID, showingDate, startTime</u>, movieID, priceCode)

Question 7: Write a SQL statement that increases numSeats by 5 for each theater whose address ends with ‘Street’.

Answer 7:

```
UPDATE Theaters
SET numSeats = numSeats + 5
WHERE address LIKE '%Street';
```

Movies(movieID, name, year, rating, length, totalEarned)
Theaters(theaterID, address, numSeats)
Showings(theaterID, showingDate, startTime, movieID, priceCode)

Question 8: Write a SQL statement that outputs the movieID and startTime for Showings whose showingDate is January 28, 2019. Your result tuples should appear sorted, with the biggest movieID first and the smallest movieID last. If movieID is the same for two result tuples, then the result tuple that has the earliest startTime should appear first. No duplicates should appear in your result.

Answer 8:

SELECT DISTINCT movieID, startTime
FROM Showings
WHERE showingDate = **DATE** '28-01-2019'
ORDER BY movieID **DESC**, startTime **ASC**;

Okay to leave out **ASC**, since **ASC** is the default.
Okay to write **DATE** '01/28/19', instead of **DATE** '28-01-2019'.

Need DISTINCT, since there could be multiple showings on January 28, 2019 that start at the same time.

Part III: (20 points, 4 each)

The following **TRUE** or **FALSE** questions refer to the Customers table that is created by the following statement:

```
CREATE TABLE Customers (
        cid             INTEGER,
        name            VARCHAR(20) UNIQUE,
        type            VARCHAR(20),
        level           VARCHAR (20) DEFAULT('Beginner') NOT NULL,
        age             INTEGER,
        PRIMARY KEY (cid),
        CHECK (age >= 18 AND age <= 65)
        );
```

Answer each question TRUE or FALSE. No explanation is required, and no part credit will be given.

Question 9: TRUE or **FALSE:** The following two SQL queries are equivalent.

```
SELECT COUNT(name)                     SELECT COUNT(DISTINCT name)
FROM Customers;                        FROM Customers;
```

Answer 9: ___**TRUE**____ Both count non-NULL values, and name is UNIQUE

Question 10: TRUE or **FALSE:** The following is a legal SQL query.

```
SELECT type, COUNT(*), AVG(age)
FROM Customers
WHERE level <> 'Advanced'
GROUP BY type;
```

Answer 10: ___**TRUE**____

Question 11: TRUE or **FALSE**: Suppose that there are no tuples in Customers. Execution of the following statement will result in an error.

```
INSERT INTO Customers(cid, name, age)
        VALUES (36, 'Cho', 50);
```

Answer 11: ___FALSE_____ level will be default, type will be NULL

Question 12: TRUE or **FALSE** : The following two queries are equivalent.

```
SELECT C.cid
FROM Customers C
WHERE C.level = ANY ( SELECT C2.level
                      FROM Customers C2
                      WHERE C2.type = 'snowboard' );
```

```
SELECT C.cid
FROM Customers C
WHERE C.level IN ( SELECT C2.level
                   FROM Customers C2
                   WHERE C2.type = 'snowboard' );
```

Answer 12: ____TRUE_____

Question 13: TRUE or **FALSE** : The CHECK condition on age in the CREATE statement would be violated if the value of age was NULL, or if it was less than 18, or if it was greater than 65.

Answer 13: ___FALSE_____ UNKNOWN when age is NULL, which is okay.

Part IV: (30 points, 10 each): The questions in Part IV ask you to write SQL statements using the tables shown below, which are 3 of the tables in our Lab Assignments.

The primary key in each table is shown underlined. Assume that there aren't any NOT NULL or UNIQUE constraints specified for these tables. You may assume the same Referential Integrity constraints that we had in our Lab Assignment tables. Data types aren't shown to keep thing simple. There aren't any trick questions about data types.

Movies(<u>movieID</u>, name, year, rating, length, totalEarned)
Theaters(<u>theaterID</u>, address, numSeats)
Showings(<u>theaterID, showingDate, startTime</u>, movieID, priceCode)

Question 14: Write a SQL statement that creates a view PoorMovies that finds the movieID and name for movies whose length is longer than 130, and which also have no showings in theaters that have 10 or more seats. No duplicates should appear in your result.

Answer 14: Here are some correct ways to create this view.

```
CREATE VIEW PoorMovies AS
        SELECT m.movieID, m.name
        FROM Movies m
        WHERE m.length > 130
        AND m.movieID NOT IN
              ( SELECT s.movieID
                FROM Showings s, Theaters t
                WHERE s.theaterID = t.theaterID
                    AND t.numSeats >= 10 );
```

DISTINCT is not necessary, since there can't be two movies with the same movieID.

Could use table names instead of tuple variables. For outer query, don't need table names or tuples variables, but somehow need to distinguish theaterID in Showings from theaterID in Theaters.

Could also write this using "<> ALL" (or "!= ALL") instead of "NOT IN".

```
CREATE VIEW PoorMovies AS
        SELECT m.movieID, m.name
        FROM Movies m
        WHERE m.length > 130
        AND NOT EXISTS
              ( SELECT *
                FROM Showings s, Theaters t
                WHERE s.movieID = m.movieID
                    AND s.theaterID = t.theaterID
                    AND t.numSeats >= 10 );
```

Same comments as for previous version, but also need to distinguish movieID in Showings from movieID in Movies.

Movies(movieID, name, year, rating, length, totalEarned)
Theaters(theaterID, address, numSeats)
Showings(theaterID, showingDate, startTime, movieID, priceCode)

Question 15: totalEarned is an attribute of Movies. Write a SQL statement that for each showingDate finds the minimum totalEarned value for the movies that have Showings on that showingDate. But don't include a result tuple for a showingDate unless the number of Showings on that showingDate is more than 5. The attributes in your result should appear as theDate and minTotalEarned. No duplicates should appear in your result.

Answer 15:

```
SELECT s.showingDate AS theDate, MIN(m.totalEarned) AS minTotalEarned
FROM Showings s, Movies m
WHERE s.movieID = m.movieID
GROUP BY s.showingDate
HAVING COUNT (*) > 5;
```

Could use table names instead of tuple variables. Need to distinguish the movieID attributes in Showings from the movieID attribute in Movies, but other attributes are unambiguous.

Instead of COUNT(*) could use COUNT on any attribute that can't be NULL; the Primary Key attributes can't be NULL.

DISTINCT is not needed since there can't be more than one group for any s.showingDate value.

Movies(movieID, name, year, rating, length, totalEarned)
Theaters(theaterID, address, numSeats)
Showings(theaterID, showingDate, startTime, movieID, priceCode)

Question 16: Write a SQL statement that finds the theaterID, showingDate and startTime for each showing that has an earlier startTime than some other showing of the same movie that's on the same showingDate. No duplicates should appear in your result.

Answer 16: Here are some correct ways to write this query.

```
SELECT DISTINCT s.theaterID, s.showingDate, s.startTime
FROM Showings s, Showings s2
WHERE s.showingDate = s2.showingDate
    AND s.startTime < s2.startTime
    AND s.movieID = s2.movieID;
```

Need tuple variables, since Showings appears twice in the FROM clause.

Need DISTINCT, since there could be multiple showings of a movie on the same date that have a later startTime than the s1 showing of that movie.

```
SELECT s.theaterID, s.showingDate, s.startTime
FROM Showings s
WHERE EXISTS
        ( SELECT *
          FROM Showings
           WHERE  s.showingDate = s2.showingDate
               AND s.startTime < s2.startTime
               AND s.movieID = s2.movieID );
```

Need tuple variables to distinguish showing references in subquery.

DISTINCT is not needed, since for each showing s, we're checking once to see if there are any s2 showings of the same movie on the same date that have a later startTime than the s1 showing of that movie.

There also are correct ways to write this query using "IN" or "= ANY", but you can't just replace EXISTS with those; see Q14 for any example showing syntaxes for "NOT EXISTS" versus "NOT IN" (or "<> ALL").