EECS-3421n: Test #2

Electrical Engineering & Computer Science

Lassonde School of Engineering

York University

Family Name: _____

Given Name: _____

Student#: _____

EECS Account: _____

Instructor: Parke Godfrey
Exam Duration: 75 minutes
Term: Winter 2019
Instructions

- **rules**
  - The test is closed-note, closed-book. Use of a calculator is permitted.
- **answers**
  - Should you feel a question needs an assumption to be able to answer it, write the assumptions you need along with your answer.
  - If you need more room to write an answer, indicate where you are continuing the answer.
  - For multiple choice questions, choose *one* best answer for each of the following. There is no negative penalty for a wrong answer.
- **notation**
  - For schema, the underlined attributes indicate a table's primary key (and are, hence, not nullable). Attributes in *italics* are not nullable. Foreign keys are indicated by FK.
  - Assume *set* semantics for relational-algebra expressions.
- **points**
  - The number of points a given question is worth is marked.
  - There are five major parts worth 10 points each, for 50 points in total.

Marking Box	
1.	/10
2.	/10
3.	/10
4.	/10
5.	/10
Total	/50

Happy PI Day!

1. [10pt] **General.** *Much choice!* Multiple Choice

   Choose *one* best answer for each of the following. Each is worth one point. There is no negative penalty for a wrong answer.

   In the *rare* case that you feel a clarification to your answer is needed, write a brief clarification on the side.

   Let $|\mathbf{T}|$ denote the number of tuples in $\mathbf{T}$.

(a) [1pt] `SQL` is derived from
- **A.** the *relational algebra*.
- **B.** the *domain relational calculus*.
- **C.** the *tuple relational calculus*.
- **D.** XQuery for XML.
- **E.** *`SQL` is a programming language, not a query language. It is not derived from any of the above.*

(b) [1pt] The *relational algebra* is an algebra because
- **A.** it contains operators for arithmetic.
- **B.** its operators map from the domain of operands back into the domain.
- **C.** it allows one to order freely its operators to form expressions.
- **D.** it contains a fixed, finite number of pre-defined operators.
- **E.** *It is not really an algebra. But calling it an algebra sounds cool!*

(c) [1pt] In a relational database system, if you join (natural join) tables $\mathbf{R}$ and $\mathbf{S}$, but $\mathbf{R}$ is empty (that is, it has no tuples),
- **A.** the system reports an error.
- **B.** the answer set is an empty table.
- **C.** the answer set is the same as table $\mathbf{S}$.
- **D.** the answer set consists of just *one* row.
- **E.** an answer set is returned; however, the results are system dependent.

(d) [1pt] $\mathbf{R} \cap \mathbf{S}$ is equivalent to
- **A.** $\mathbf{R} - (\mathbf{R} - \mathbf{S})$
- **B.** $\mathbf{R} - (\mathbf{S} - \mathbf{R})$
- **C.** $(\mathbf{R} - \mathbf{S}) - \mathbf{R}$
- **D.** $-((-\mathbf{R}) \cup (-\mathbf{S}))$
- **E.** *There is not enough information to answer this.*

(e) What does the query "`select max(R.B) from R;`" return if $\mathbf{R}$ is empty?
- **A.** An empty table of one column.
- **B.** A table of one column with one row with the value ⟨NULL⟩.
- **C.** A table of one column with one row with the value ⟨INF⟩.
- **D.** An error message.
- **E.** *Not enough information to determine.*

(f) [1pt] In *relational algebra*, we can write any expression using *join* ("$\bowtie$") using the following instead.

A. π, $\cap$
B. π, $\cup$
C. π, σ, $\times$
D. π, $-$
E. *Join cannot be rewritten with any combination of the other R.A. operators.*

(g) [1pt] Which of the following may evaluate to a different answer than $\mathbf{R} \bowtie (\mathbf{S} \bowtie \mathbf{T})$?

A. $(\mathbf{R} \bowtie \mathbf{S}) \bowtie \mathbf{T}$
B. $\mathbf{R} \bowtie (\mathbf{T} \bowtie \mathbf{R})$
C. $(\mathbf{S} \bowtie \mathbf{T}) \bowtie \mathbf{R}$
D. $\mathbf{T} \bowtie (\mathbf{S} \bowtie \mathbf{R})$
E. *They all evaluate the same.*

(h) [1pt] Consider the schema

$\mathbf{R}(\underline{\mathsf{A}}, B)$ FK (B) refs $\mathbf{R}$ (A)

What is the *largest* that $|\mathbf{R} \bowtie \pi_{\mathsf{A}\to\mathsf{B},\mathsf{B}\to\mathsf{A}}(\mathbf{R})|$ can be?

A. 0
B. $|\mathbf{R}|$
C. $\frac{1}{2}|\mathbf{R}|$
D. $2|\mathbf{R}|$
E. $|\mathbf{R}|^2$

(i) [1pt] Consider table $\mathbf{R}(\underline{\mathsf{A}}, \mathsf{B})$ for which B is of type *integer* and $|\mathbf{R}| = n > 0$. How many tuples will the query

```
select A from R where B <= 13 or B > 13;
```

return?

A. 0
B. $\frac{1}{2}n$
C. n
D. n^2
E. *There is not enough information to answer this.*

(j) [1pt] Consider the relations $\mathbf{R}(\underline{\mathsf{A}}, \underline{\mathsf{B}})$, $\mathbf{S}(\underline{\mathsf{B}}, \underline{\mathsf{C}})$, and $\mathbf{T}(\underline{\mathsf{C}}, \underline{\mathsf{A}})$.
One of these is not like the others. That is, one can evaluate differently than the other four. Which one?

A. $\pi_{\mathsf{A},\mathsf{B}}((\mathbf{R} \bowtie \mathbf{S}) \bowtie \mathbf{T})$
B. $\pi_{\mathsf{A},\mathsf{B}}(\mathbf{R} \bowtie (\mathbf{T} \bowtie \mathbf{S}))$
C. $\mathbf{R} \bowtie \pi_{\mathsf{A},\mathsf{B}}(\mathbf{S} \bowtie \mathbf{T})$
D. $\pi_{\mathsf{A},\mathsf{B}}(\mathbf{R} \bowtie \mathbf{T}) \bowtie \pi_{\mathsf{A},\mathsf{B}}(\mathbf{R} \bowtie \mathbf{S})$
E. $\pi_{\mathsf{A},\mathsf{B}}(\mathbf{R} \bowtie \mathbf{S}) \bowtie \pi_{\mathsf{A},\mathsf{B}}(\mathbf{S} \bowtie \mathbf{T})$

---

2. [10pt] **Relational Algebra.** *Riddle me this!* SHORT ANSWER

For Questions 2a to 2d, use the *Colours* schema in Figure 1 on page 13 (as used in examples in class) to write *relational-algebra* queries for the English questions posed and vice versa.

(a) [2pt] Show products by prod# and pname that come in colour *orange*.

(b) [2pt] Show products by prod# and pname that are owned by at least two customers.

(c) [2pt] Show pairs of customers—*first* (cust#) and *fname* (cname) for the first customer and *second* (cust#) and *sname* (cname) for the second customer—such that the second customer owns an item in the first customer's favourite colour.

(d) [2pt] Consider the following R.A. expression.

$$\pi_{\mathsf{colour}}(\pi_{\mathsf{colour,prod\#,name}}(\mathbf{Product} \bowtie \mathbf{Avail\_colour}) - \pi_{\mathsf{colour,prod\#,name}}(\sigma_{\mathsf{cost}>\mathsf{cost2}}(\pi_{\mathsf{colour,prod\#,name,cost}}(\mathbf{Product} \bowtie \mathbf{Avail\_colour}) \bowtie \pi_{\mathsf{colour,prod\#,name,cost}\to\mathsf{cost2}}(\mathbf{Product} \bowtie \mathbf{Avail\_colour}))))$$

State what the query asks in English.

Note that you will get *zero* credit if you use database terms in your answer! (E.g., "Well, the query first *joins* two tables, taking the *projection* of…" does not count!)

(e) [2pt] Consider the following `SQL` query.

```
select distinct Z.A, X.C
from R Z, S X
where Z.B not in (
        select Y.B
        from S Y
        where X.C = Y.C);
```

Write an equivalent R.A. expression for it.

3. (10 points) **Queries in SQL.** *Ask me anything.* Exercise

Consider the *Movie* schema in Figure 2 on page 13 for Questions 3a to 3d.

(a) [3pt] Write an SQL query that answers the following.
Report *directors* by p# and name with the movies that they have directed by title, studio, year, and genre.
Order by name, p#, year, studio, title (all *ascending*).

(b) [2pt] Write an SQL query that answers the following.
Report *directors* by p# and name with the number of movies that they have directed as #movies. You may assume that every director has directed some movie.
Order by name, p# (both *ascending*).

(c) [3pt] Consider the following SQL query.

```
select  P.p#, P.name,
        ( select count(*)
          from Cast C
          where P.p# = C.p#
        ) as #roles
from Person P;
```

Assume that everyone who has been cast in a movie is an *actor*, and that every *actor* has been in some movie.

Write an SQL query that means the same thing, but that uses *no* nested (sub-) queries.

(d) [2pt] Consider the following SQL query.

```
select P.name, P.gender, C.role, C.minutes, M.title, M.studio, M.year
from Person P, Cast C, Movie M, Authored A, Person W
where C.actor = P.p#
  and C.title = M.title and C.studio = M.studio and C.year = M.year
  and M.genre = 'SciFi'
  and A.title = M.title and A.year = M.year
  and A.writer = W.p#
  and W.name = 'Hampton Fancher';
```

State what the query asks in English.

Note that you will get *zero* credit if you use database terms in your answer! (E.g., "Well, the query first *joins* two tables, taking the *projection* of..." does not count!)

4. (10 points) **Query Logic.** *Fascinating.* Analysis

(a) [2pt] Consider the schema

S(B, *C*)
R(A, *B*)
FK (B) refs **S**(B)

What is $|\mathbf{R} \bowtie \mathbf{S}|$? (That is, what is the cardinality of $\mathbf{R} \bowtie \mathbf{S}$?)
Give your answer in the simplest form.

(b) [2pt] You execute the following command against the database system for the *Colours* database (schema in Figure 1 on page 13).

```
insert into Item (item#, prod#, cust#, colour, date_sold)
values
    (1729, 23, 13, 'hot pink', '03/14/2019');
```

Is this tuple now guaranteed to have been added to the table **Item**?
Why or why not?

(c) [2pt] Say that "**R**.A <> **S**.B" appears as a predicate in the *where* clause of an SQL query. To what does the predicate evaluate when **R**.A is *null* and **S**.B is *null*?

(d) [2pt] Consider the *Colours* database (schema in Figure 1 on page 13) and the following SQL query.

```
select distinct C.cust#, P.prod#
from Customer C, Avail_Colours A
where C.fav_colour <> A.colour;
```

State what the query asks in English.

Note that you will get *zero* credit if you use database terms in your answer! (E.g., "Well, the query first *joins* two tables, taking the *projection* of..." does not count!)

(e) [2pt] Consider the following SQL query.

```
select A, B
from (values
        (3.1416, 'x'),
        (2.7183, 'y')
     ) as R (A, B)
except
select C
from (values
        (3.1416)
     ) as S (C);
```

How does the query evaluate?

5. [10pt] **The SQL Language.** *Speaking quite logically.* Short Answer

Consider the following for Questions 5a, 5b, and 5c.

Dr. Mark Dogfury has written his own `SQL` database system that he calls **WoofSQL**. Unfortunately, he forgot to implement "distinct", and he forgot to implement "having"! However, he did implement the rest of `SQL`.

(a) [3pt] Can you still write queries in **WoofSQL** that eliminate duplicate values, despite missing "distinct"?

Why or why not?

(b) [2pt] Can you still write all the queries that you can in regular, *full* `SQL` in **WoofSQL**, even though you do not have the *having* clause?

Why or why not?

(c) [2pt] Dr. Dogfury implemented *distinct* and *having* for **WoofSQL v2**, but somehow the *intersect* operator got left out of **v2**!

Can you still write all the queries that you can in regular, *full* `SQL` in **WoofSQL v2**, even though you do not have the *intersect* operator?

Why or why not?

(d) [3pt] Say that you know that the **Product** table in the *Colours* database (schema in Figure 1 on page 13) has 45 tuples in it.

How many tuples are returned by the following query?

```
select prod#, pname
from Product as P,
     (values ('A'), ('B'), ('C'))
        as Q (X);
```

Extra Space

Relax. Turn in your test. Return to the wild! Eat some pie!

Reference *(Detach this page for convenience, if you want.)*

Schema for the Colours Database.

Customer	
cust#	PK
cname	
fav_colour	
phone#	

Item	
item#	PK
prod#	FK to **Product**
cust#	FK to **Customer**
colour	
date_sold	

Product	
prod#	PK
pname	
cost	
maker	FK to **Company**

Avail_Colours	
prod#	PK, FK to **Product**
colour	PK

Figure 1: Colours Schema.

Schema for the Movie Database.

Person(p#, *name*, birthdate, *nationality*, *gender*)
Actor(p#, *aguild#*)
  FK (p#) refs **Person**
Director(p#, *dguild#*)
  FK (p#) refs **Person**
Writer(p#, *wguild#*)
  FK (p#) refs **Person**
Studio(name)
ScreenPlay(title, year)
Authored(title, year, writer)
  FK (title, year) refs **ScreenPlay**
  FK (writer) refs **Writer** (p#)
Movie(title, studio, year, *genre*, *director*, *length*)
  FK (studio) refs **Studio** (name)
  FK (title, year) refs **ScreenPlay**
  FK (director) refs **Director** (p#)
Cast(title, studio, year, role, actor, *minutes*)
  FK (title, studio, year) refs **Movie**
  FK (actor) refs **Actor** (p#)
Affiliated(director, studio)
  FK (director) refs **Director** (p#)
  FK (studio) refs **Studio** (name)

Figure 2: Movie Schema.

Reference

The Relational-Algebra Operators.

$\boldsymbol{\sigma}$: selection
$\boldsymbol{\pi}$: projection
$\bowtie$: join
$\times$: cross product
$\cup$: union
$\cap$: intersection
$-$: difference
$\boldsymbol{\rho}$: rename